Mini-Curso sobre Banco de Dados: Firebird - Sua história e características - Parte 1

Clube Delphi

Efeitos Gráficos
Melhorando a aparência de aplicações Delphi

Ano 12 :: Edição 135 :: R$ 11,90

DevMedia

DELPHI XE 2

Fique por dentro do há de novo

Boas Práticas
Um estudo sobre Scrum e XP

Telnet e Expressões Regulares
Exemplo prático de utilização

ISSN 1517990-7

Somos tão apaixonados por tecnologia que o nome da empresa

diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,

assim como você.

Estrutura

100% NACIONAL. Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única. Conheça!

Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce | Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486

Sumário

Delphi

Boas Práticas

06 – Melhorando a aparência de aplicações Delphi
[*Jederson Zuchi e Giuliano Scombatti Pintos*]

Mini-Curso /Banco de Dados

16 – Firebird: Poderoso, Leve, Gratuito e Open-Source
[*Fabiano Moura*]

Boas Práticas

24 – Delphi XE2, uma visão geral no que há de novo
[*Fabrício Hissao Kawata*]

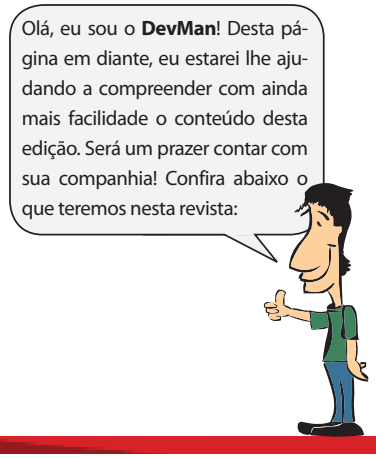
Delphi Avançado

34 – Telnet e Expressões Regulares
[*Felipe Monteiro de Carvalho*]

Boas Práticas

Boas Práticas

44 – Um estudo sobre metodologias ágeis
[*Fabrício Hissao Kawata*]



Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

Clube Delphi

Ano 12 - 135ª Edição 2011 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral

Paulo Quicoli
pauloquicoli@devmedia.com.br

Editor Técnico

Leandro Daniel
contato@leandrodaniel.com

Equipe Editorial

Vitor Luiz Rubio
vitorrubio@gmail.com

Rafael Stavarengo
faelsta@gmail.com

Jornalista Responsável

Kaline Dolabella - JP24185

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

www.devmedia.com.br/mancad
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Paulo Quicoli - Editor da Revista
pauloquicoli@gmail.com

EDITORIAL

Esta edição tem duas novidades. A primeira e mais importante: Delphi XE 2. Nosso amigo Fabrício expõe de forma pontual os novos recursos do IDE, destacando o novo framework visual FireMonkey e a nova forma de ligação de dados, o LiveBinding. Quem acreditava que o Delphi estava com seus dias contados vai se surpreender com o que foi feito pela Embacadero nesta versão. Temos 64 bits, desenvolvimento para dispositivos móveis. Android e iOS que se preparem para os programadores Delphi!

A segunda novidade é que estou assumindo a edição geral da revista Clube Delphi. Acompanho a revista desde suas primeiras edições, vendo de perto sua evolução e legado para toda a comunidade Delphi do Brasil. A figura do Guinther se tornou sinônimo de Clube Delphi, e quero assumir perante vocês o compromisso de manter a mesma qualidade deixada por ele e por toda a equipe. Como disse, sou leitor há tempos. Em um determinado dia fui convidado a colaborar com a revista. Envieis artigos para revista impressa e para o site, vídeo aulas, e até mesmo no fórum participava. Fui aprendendo como escrever de forma clara e como colaborar mais. Nisso fui convidado a fazer parte da equipe editorial, como revisor técnico da Clube Delphi. Aprendi muito nesse período até que me tornei também revisor técnico da .NET Magazine. E para fechar o ciclo, hoje assumo a edição geral da Clube Delphi.

Continuando, se você ainda utiliza o Delphi 7, ou ainda não migrou para o XE 2, veja como melhorar visualmente sua aplicação utilizando recursos do próprio sistema operacional, sem a necessidade de componentes de terceiros. O Jederson Zuchi e o Giuliano Scombatti mostram de forma simples e prática como fazer uso disso. Em outro artigo do Fabrício Kawata vamos compreender e aprender sobre metodologias ágeis. Não existe apenas o Scrum, há também o XP. Em seu artigo temos um estudo sobre os dois modelos.

Ainda nesta edição temos um mini-curso do Fabiano Moura que irá explorar o Firebird, começando com sua história e recursos. É uma ótima oportunidade para novos desenvolvedores que talvez só conheçam o SQL Server de abrir seus horizontes. E se seu nível de programação é o avançado, vai gostar do artigo do Felipe Monteiro, sobre Telnet e Expressões Regulares.

Para encerrar, quero dizer que estou disponível para ouvir você leitor, suas dúvidas e sugestões, e até porque não dizer, reclamações. Um abraço e boa leitura!

Paulo Quicoli
pauloquicoli@gmail.com
Editor Geral
quicoli.wordpress.com
twitter.com/pauloquicoli

ClubeDelphi PLUS

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

DevMedia

Sistema de créditos DevMedia

Conteúdo **completo** ao seu alcance

Com o sistema de créditos DevMedia você pode comprar individualmente mais de 4.000 vídeo aulas, 8000 artigos, além de palestras e vídeos com formações completas.

TODAS as vídeo aulas que você sempre quis ver a partir de

R$ 1,20

ECONOMIA: você não precisa assinar 8 produtos diferentes para ter acesso ao conteúdo completo do site!

Saiba mais sobre o Sistema de Créditos!

www.devmedia.com.br/creditos

DevMedia

Efeitos Gráficos
Melhorando a aparência das apli

Resumo DevMan

De que trata o artigo:

Este artigo demonstra que é possível criar alguns efeitos visuais de uma maneira bem simples para serem utilizados em aplicações desktop utilizando a VCL sem ter que recorrer a outras tecnologias.

Em que situação o tema é útil:

Permite que o desenvolvedor personalize sua aplicação, saindo um pouco da tradicional e comportada interface gráfica de sempre dos formulários e seus componentes, criando efeitos, mensagens e aplicando skins. Este tema é bastante útil para quem quer alterar o estilo visual da sua aplicação e não tem ideia de como fazer ou por onde começar, bem como dá algumas noções para que o usuário possa desenvolver seus próprios efeitos e aplicar skins em outros componentes.

Efeitos Gráficos:

A qualidade de software é um tema hoje muito comum, e todas as empresas têm buscado aprender técnicas para obter essa melhoria. Qualidade é indiscutível, todo software precisa ter. Por ser um ponto comum, e assumindo que as aplicações a possuem, o que pode diferenciar o seu sistema em relação ao do concorrente? Recursos e funcionalidades. E se ambos forem "parecidos"? Com certeza será a experiência do usuário e sua aparência o diferencial. Neste artigo vamos explorar o quesito aparência e mostrar como personalizar a interface de aplicações sem utilizar biblioteca de terceiros.

Atualmente com a popularização de diversas tecnologias que permitem criar aplicações desktop com efeitos gráficos e animações, como a WPF (Windows Presentation Foundation) e a Adobe AIR, é comum recorrer a tais ferramentas quando se faz necessário o desenvolvimento de uma aplicação multimídia ou simplesmente para deixar uma aplicação com um "ar mais amigável" e fugir dos seus tradicionais e simples efeitos. Porém, vale a pena ressaltar que é possível desenvolver com Delphi funções e procedimentos, simples ou complexos, voltados para criar efeitos gráficos que poderão ser utilizados em toda a aplicação, em seus formulários e componentes. Efeitos que possam incrementar e até mesmo competir com as tecnologias descritas anteriormente. Também é importante falar da OpenGL, disponível inclusive para o Delphi, que permite incrementar ainda mais o ambiente gráfico da aplicação, disponibilizando assim a criação de objetos, efeitos ou cenas em 3D na mesma.

Talvez você não saiba, mas uma nova API gráfica foi adicionada no Windows 7. Esta nova API é chamada de Direct2D, para desenvolvimento de gráficos 2D no Windows 7. Seu objetivo é substituir as antigas APIs GDI e GDI+ que são mais lentas. O GDI, ou Graphics Device Interface, é um dos três subsistemas principais do Microsoft Windows. É um padrão desse sistema operacional para representar objetos gráficos e transmitilos para dispositivos de saída, como monitores e impressoras. É responsável por tarefas como desenhar linhas, curvas e fontes, providenciando todo um API específico para a execução dessas operações. No Delphi a classe TCanvas é responsável por consumir esses recursos da API. A GDI+ é a sucessora da GDI, foi

cações com Delphi

Jederson Zuchi e Giuliano Scombatti Pinto

introduzida no Windows XP e trouxe grandes melhorias, como antialiasing de gráficos 2D, criação de gradientes e suporte muito melhor aos formatos de imagem como o JPEG e o PNG (inclusive foi na GDI+ que ocorreu a famosa falha que poderia fazer com que a exibição de uma simples imagem JPEG na internet permitisse a instalação de um malware no sistema afetado). Hoje a GDI e a GDI+ são responsáveis pelos efeitos 2D do Windows. A GDI + trouxe melhorias nas transparências e isso pode ser visto na interface do Windows XP. Porém continuamos a ter alguns problemas como falta de recursos mais avançados e desempenho uma vez que a GDI não é acelerada pela GPU.

A função AnimateWindow

Através do uso da função AnimateWindow, presente na Unit Windows, é possível criar um simples efeito gráfico animado quando uma janela (formulário) é exibida ou ocultada, podendo conter os efeitos de rolar, deslizar, contrair, expandir ou de transparência. Para utilizar esta função, além de adicionar a Unit Windows, é necessário informar um "handle", que permite que uma janela ou um dispositivo de contexto seja referenciado por uma aplicação Windows e posteriormente informar qual o efeito desejado, bem como sua duração.

Parâmetro	Descrição
HWND: Handle	Handle do formulário que será animado, exemplo: TForm1.Handle.
DWTIME: DWORD	O tempo em milisegundos que a animação vai durar.
FLAGS: DWORD	O tipo da animação.

Tabela 1. Parâmetros da função AnimateWindow()

Nota do DevMan

O sistema operacional Windows possui vários objetos visuais, podem - e são - instanciados pelos mais diversos aplicativos para exibir informações. Mesmo em Delphi estamos instanciando objetos do Windows. Por exemplo, controles que herdam de TWinControl encapsulam em si com controle do sistema operacional. Mas como o Windows identifica todos esses objetos durante sua execução? Esses controles possuem um identificador chamado Handle. Ele é utilizado para essa finalidade.

O evento OnCreate é executado no momento em que um formulário é criado, sendo assim, basta adicionar a função Animate Window com seus parâmetros neste evento para criar um efeito de exibição. Se o desenvolvedor preferir, pode adicionar o mesmo no evento OnShow, executado quando o formulário for exibido. Para criar um efeito no momento em que o formulário for fechado pode-se utilizar o evento OnClose.

Sintaxe da função AnimateWindow

A sintaxe da função AnimateWindow() é a seguinte: function AnimateWindow(HWND: Handle, dwTime: DWORD, dwFlags: DWORD) : Boolean. A **Tabela 1** mostra detalhadamente para que serve cada um dos parâmetros da função:

Ao utilizar esta função, o desenvolvedor deve escolher qual é o tipo da animação desejada através de flags. Vale ressaltar que flags são utilizados para informar quais opções estão habilitadas em um determinado componente e, consequentemente os que não foram informados estão desabilitados ou inativos. Sendo assim, o desenvolvedor informa quais os efeitos que serão executados, uma vez que podem ser utilizados vários ao mesmo tempo. A **Tabela 2** mostra quais flags que o desenvolvedor pode utilizar, bem como algumas regras para a utilização dos mesmos em conjunto.

Flag	Descrição
AW_ACTIVATE	Ativar o formulário. Não deve ser usado em conjunto com AW_HIDE.
AW_BLEND	Efeito de Fade-In ou Fade-Out (Transparência gradual), devendo ser usado em um formulário que esteja ativo.
AW_CENTER	Faz que um formulário se contraia gradualmente em conjunto com AW_HIDE ou que ele se expanda caso AW_HIDE não seja utilizado.
AW_HIDE	Ocultar o formulário.
AW_HOR_POSITIVE	Anima o formulário da Esquerda para a Direita, em um efeito de rolar ou deslizar, devendo ser ignorada ao utilizar AW_CENTER ou AW_BLEND.
AW_HOR_NEGATIVE	Anima o formulário da Direita para a Esquerda, em um efeito de rolar ou deslizar, devendo ser ignorada ao utilizar AW_CENTER ou AW_BLEND.
AW_SLIDE	Efeito de Deslizar o formulário, sendo ignorado se AW_CENTER estiver sendo utilizado.
AW_VER_POSITIVE	Anima o formulário de Cima para Baixo, em um efeito de rolar ou deslizar, devendo ser ignorada quando ao AW_CENTER ou AW_BLEND.
AW_VER_NEGATIVE	Anima o formulário de Baixo para Cima, em um efeito de rolar ou deslizar, devendo ser ignorada quando ao AW_CENTER ou AW_BLEND.

Tabela 2. Flags utilizados na função AnimateWindow()

Como observado, o desenvolvedor pode utilizar AW_HOR_POSITIVE ou AW_HOR_NEGATIVE em conjunto com AW_VER_POSITIVE ou AW_VER_NEGATIVE para obter um efeito na diagonal. Para concluir, a função AnimateWindows() retorna um valor do tipo boolean, ou seja, se foi executada com êxito retorna o valor verdadeiro senão retorna falso. Para demonstrar temos a **Listagem 1** um efeito aplicado no momento de sua exibição.

Pode-se entender os parâmetros do evento AnimateWindow da seguinte forma: o primeiro parâmetro é o handle do próprio formulário, o segundo parâmetro é o tempo em milisegundos que o efeito e sua animação irão durar, ou seja 1 segundo. O último contém os tipos da animação, ativar e expandir do centro. Para realizer uma animação diagonal é muito fácil. É uma troca de parâmetros como mostra a **Listagem 2**.

Listagem 1. Expandir o formulário e centralizá-lo manualmente

```
procedure TFrPrincipal.FormShow(Sender: TObject);
begin
 {Centralizar o Formulário Manualmente}
 Self.Left := (Screen.Width - Self.Width) div 2;
 Self.Top := (Screen.Height - Self.Height - 30) div 2;
 AnimateWindow(Self.Handle, 1000, AW_ACTIVATE or AW_CENTER);
end;
```

Listagem 2. Deslizando o formulário na diagonal

```
procedure TFrPrincipal.FormShow(Sender: TObject);
begin
 {Centralizar o Formulário Manualmente}
  Self.Left := (Screen.Width - Self.Width) div 2;
 Self.Top := (Screen.Height - Self.Height - 30) div 2;
 AnimateWindow(Self.Handle, 1000, AW_ACTIVATE or AW_SLIDE or AW_HOR_
 POSITIVE or AW_VER_POSITIVE);
end;
```

O efeito que faz com que o formulário deslize é bem simples e não requer muito código, desta forma é possível incrementar os efeitos dos formulários do sistema, deixando a aplicação mais rica e mais usual para o usuário final, uma vez que "quebra" o visual parado da abertura dos formulários.

Efeito de fade-in e fade-out em mensagens ou conteúdos

Este efeito é atualmente muito conhecido nas páginas web e será simulado na aplicação Delphi para substituir as tradicionais mensagens de aviso ou pergunta obtidas com as funções ShowMessage() ou MessageDlg(). Consiste em criar um efeito fade-in (transparência gradual que evolui em um determinado tempo até exibir um conteúdo) para exibir um formulário com a mensagem e um efeito fade-out (transparência gradual que evolui em um determinado tempo até ocultar um conteúdo) quando o usuário clicar em um determinado botão, fechando assim o formulário contendo a mensagem de uma maneira bem simples. Neste exemplo serão criados dois formulários em tempo de execução, um que receberá o efeito de fade e bloqueará todo o conteúdo restante da tela e o outro que vai simular uma caixa de diálogo centralizada. Também deverão ser criados em tempo de execução os objetos e eventos. O primeiro passo é a criação de um procedimento de objeto para ser utilizado em tempo de execução. Em seguida, é criada uma classe que conterá tal evento e posteriormente a função Mensagem que será responsável por exibir, ocultar e retornar um valor em ModalResult, assim como ShowMessage().

A função Mensagem() deverá conter alguns parâmetros para serem utilizados em diversos tipos de mensagem. O primeiro parâmetro pode ser a mensagem que se deseja mostrar, o próximo, o título da mesma e para concluir, seu estilo. De acordo com o que for informado (mensagem de aviso, pergunta ou erro), deverá exibir a imagem apropriada e seus botões. A sintaxe da função é a seguinte: function Mensagem(TextoMsg, Titulo, Tipo: String) : TModalResult;

O trecho de código da **Listagem 3** mostra como criar o procedimento que será utilizado em tempo de execução, bem como uma classe que terá tal procedimento.

Procedimento OnClickSetModalResult

Este procedimento foi criado para que possa ser utilizado em tempo de execução e, como seu próprio nome sugere, no momento em que o usuário clicar em um dos botões do formulário que contém a mensagem, a propriedade ModalResult de tal formulário será alterada de acordo com o botão clicado. Vale ressaltar que ModalResult é o mesmo resultado retornado pela função ShowModal(). Para descobrir qual foi a opção do usuário no formulário de mensagem, basta fazer da mesma forma como é feita com ShowModal(), comparando se o valor é igual a "mrYes", "mrNo", "mrOk", ou ainda, 6, 7, entre outros. Na **Listagem 4** segue o trecho de código responsável pelo procedimento.

Listagem 3. Criando o código para ser utilizado em tempo de execução

```
type
 TEvento = procedure(Sender: TObject) of object;
 TObjeto = class
 private
  FEvento: TEvento;
 public
  property Evento: TEvento read FEvento write FEvento;
  class procedure OnClickSetModalResult(Sender: TObject);
 end;
```

Listagem 4. Setando a propriedade ModalResult do formulário em tempo de execução

```
class procedure TObjeto.OnClickSetModalResult(Sender: TObject);
begin
 if (TSpeedButton(Sender).Name = 'BtSim') then
  TForm(TSpeedButton(Sender).Parent).ModalResult := mrYes
 else if (TSpeedButton(Sender).Name = 'BtNao') then
  TForm(TSpeedButton(Sender).Parent).ModalResult := mrNo
 else if (TSpeedButton(Sender).Name = 'BtOk') then
  TForm(TSpeedButton(Sender).Parent).ModalResult := mrOk;
end;
```

Foram utilizadas as classes do formulário e do botão para generalizar seu uso, ou seja, para que qualquer botão (neste caso do tipo TSpeedButton) pudesse utilizar este procedimento e se referir

ao seu formulário. Neste exemplo, o TSpeedButton foi utilizado por padrão pelas suas particularidades gráficas, como exemplo a propriedade Flat que torna o botão transparente, porém um outro tipo de botão poderia ser utilizado.Vale a pena lembrar que, por padrão, o TSpeedButton não possui a propriedade ModalResult, sendo assim, definimos o resultado em tempo de execução direto no formulário.

O trecho de código TSpeedButton(Sender).Parent se refere ao formulário "pai" do botão que foi clicado pelo usuário.

Nota do DevMan

A orientação a objetos introduz conceitos que podem representar o domínio de um problema de forma mais eficiente. Um desses conceitos é a generalização de tipos, ou somente herança. É possível criar uma hierarquia de classes (tipos) que herdam comportamento e dados de seus ancestrais. Para ilustrar como isso pode ser prático imagine a situação de um sistema comercial onde é necessário se ter pessoas jurídicas e pessoas físicas. Aplicando a orientação a objetos nesse cenário é possível ter uma classe Pessoa, que possua o que há de comum entre Física e Jurídica. Por exemplo: Nome, Endereço, Telefones, Emails. E então especializamos Pessoa em Física com propriedades como nascimento, sexo, cpf. Depois especializamos novamente Pessoa em Jurídica, agora apresentando propriedades como razão social, cnpj, sócios e assim por diante. Apesar de ser um dos conceitos mais fáceis de entender, é o que deve ser utilizado com maior cautela, pois a criação e longas hierarquias não é uma boa prática.

Função mensagem

A função que será responsável pela exibição das informações, ou seja, da mensagem em si e seus objetos possui a seguinte sintaxe: function Mensagem(TextoDaMensagem, Titulo, Tipo: String): TModalResult;

Para o funcionamento desejável da função mensagem, deverão ser criados dois formulários em tempo de execução através do procedimento CreateForm(). No formulário de fundo que exibirá o efeito de fade-in e fade-out, deverão ser alteradas as propriedades AlphaBlend e AlphaBlendValue. A propriedade AlphaBlend permite adicionar o efeito de transparência devendo então ser alterada para True. Já a propriedade AlphaBlendValue define o valor do filtro Alpha, onde 0 é o menor valor, ou invisível, e 255 o máximo, totalmente visível. Sendo assim, valores intermediários criarão um efeito de transparência. Também será necessária a alteração de outras propriedades como a cor de fundo do formulário para preta, nenhum estilo de borda, maximizado e desativado. No outro formulário que conterá a mensagem, basta deixar seu posicionamento centralizado, e se for de preferência do desenvolverdor, ele poderá colocar uma imagem de fundo no formuário de mensagem, ou ainda remover o estilo de borda e mudar a cor de fundo para criar uma janela personalizada, onde pode ser criado um TShape para desenhar um retângulo e simular uma barra de título e outro TShape para o restante do formulário, como mostra a **Figura 1**.

O exemplo da **Listagem 5** demonstra como criar dois formulários em tempo de execução e definir algumas de suas propriedades comentadas anteriormente.

Nota do DevMan

Quando se instancia dinamicamente um formulário em Delphi, em seu construtor é esperado um parâmetro chamado Owner, do tipo TComponent. Isto porque todo controle que herda de TControl exige isso. Uma dúvida dos desenvolvedores menos experientes está em saber o que passar nesse parâmetro. Para saber o que passar é preciso entender primeiramente qual a finalidade desse parâmetro.

AOwner representa o objeto que será o proprietário da instância do formulário a ser criado. Ele será responsável por liberar a memória ocupada pelo formulário e esse formulário também será incluído no array Components de AOwner. Quando um objeto proprietário é destruído, os objetos que ele detém também são destruídos. Isso é muito importante para formulários e varia de acordo com o que é passado. Comumente três opções são utilizadas:

Nil – significa que o formulário a ser criado não terá proprietário algum, devendo ele mesmo ser responsável por sua liberação de memória através de método Free.

Self – o objeto proprietário será o que está criando o formulário. Este caso fica mais claro com um exemplo. Se você inserir no formulário principal da sua aplicação um botão e nele instanciar um formulário como a seguir: form := TForm2.Create(Self), este self se refere ao formulário principal.

Application – utiliza a variável global do tipo TApplication que é criada quando a aplicação entrou em execução.

Existem algumas recomendações. Se o formulário a ser criado será destruído na sequência, passe sempre nil em seu construtor. Assim você mesmo se responsabiliza pela liberação de recursos. Mas, porque não utilizar Application nesses casos? Uma aplicação pode conter centenas de formulários, e neles, centenas de controles. Ao passar Application como parâmetro, uma notificação é enviada a cada componente e formulário possuído por Application. Isso pode gerar um delay na apresentação do formulário. A utilização de Self é indicada quando o formulário a ser criado não for Modal e não está sendo criado a partir de um "menu principal", isso porque se o proprietário for fechado, o formulário será liberado da memória

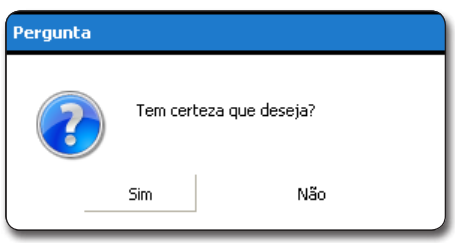


Figura 1. Formulário da função mensagem

Após criar os formulários, os objetos também deverão ser criados em tempo de execução. Primeiramente, podem ser criados dois rótulos (TLabel), um para o título da janela que deverá ser posicionado no topo do formulário, e o outro que conterá a mensagem posicionado no centro. O código a seguir mostra como criar um TLabel e através da propriedade Parent, associá-lo ao formulário, neste caso, pertencendo ao formulário FrMsg:

```
LbMensagem := TLabel.Create(FrMsg);
with LbMensagem do
begin
  Parent := FrMsg;
  Caption := TextoDaMensagem;
  Visible := True;
end;
```

Posteriormente, deve-se criar a imagem (TImage) que conterá o símbolo da mensagem e também associá-la ao formulário, podendo ter também uma estrutura "if" para carregar a imagem de acordo com o tipo da mensagem, ou seja, se for pergunta carrega-se a imagem de um ponto de interrogação e assim sucessivamente.

Listagem 5. Criando os formulários em tempo de execução

```
Application.CreateForm(TForm, FrFundo);
Application.CreateForm(TForm, FrMsg);
with FrFundo do
begin
  AlphaBlend := True;
  AlphaBlendValue := 0;
  Color := clBlack;
  BorderStyle := bsNone;
  WindowState := wsMaximized;
  Enabled := False;
end;
with FrMsg do
begin
  Color := clWhite;
  Width := 300;
  Height := 150;
  Position := poScreenCenter;
  BorderStyle := bsNone;
  WindowState := wsNormal;
end;
```

Dando sequência, para concluir a criação de objetos no formulário basta criar os botões e também associá-los ao mesmo, como já mencionado através da propriedade Parent. Os botões podem ser do tipo TSpeedButton, já que ele possui alguns recursos gráficos úteis como a propriedade Flat, permitindo alterar seu visual. Também será necessário definir o evento OnClick dos botões para que, quando forem clicados, executem o evento responsável por definir o retorno da mensagem em ModalResult. No código que segue temos demonstração de como criar um botão e associá-lo a formulário, bem como atribuir o evento comentado anteriormente:

```
BtSim := TSpeedButton.Create(FrMsg);
BtSim.Parent := FrMsg;
BtSim.OnClick := TObjeto.OnClickSetModalResult;
```

É importante destacar que o ideal é criar uma estrutura para posicionar os componentes no formulário, ou seja, para posicionar o TLabel sempre centralizado pode-se pegar o resultado da divisão da largura do formulário por 2 e subtrair do resultado da divisão da largura do TLabel também por 2 e atribuir tal valor a propriedade Left do TLabel, uma vez que o mesmo terá tamanhos diferentes baseados na descrição da mensagem, Traduzindo para códigos é algo similar a isto:

```
LbMensagem.Left := FrMsg.Width div 2 –
  LbMensagem.Width div 2
```

Para criar o efeito animado de fade-in, comentado anteriormente, serão utilizadas as propriedades AlphaBlend e AlphaBlendValue do mesmo, também já descritas anteriormente. Para criar a animação, neste exemplo é utilizado o procedimento sleep(), da Unit SysUtils, em conjunto com um for e adicionado um código que altera o valor da propriedade AlphaBlendValue gradualmente. Vale ressaltar que o procedimento sleep() faz a aplicação aguardar um determinado tempo para continuar executando o restante do código, em outras palavras uma pausa. Sendo assim é possível controlar o tempo de evolução do efeito sem a necessidade de utilizar um outro recurso, como por exemplo um Timer, que também permite executar um certo código durante o tempo solicitado. Após terminar o laço for que criou o efeito animado, o formulário da Mensagem é exibido. O exemplo da **Listagem 6** demonstra como criar este efeito.

Listagem 6. Criando o efeito de Fade-in do formulário

```
FrFundo.Visible := True;
for i := 1 to 5 do
begin
   FrFundo.AlphaBlendValue := FrFundo.AlphaBlendValue +30;
   Sleep(1);
end;
FrMsg.ShowModal;
```

O formulário de fundo foi mostrado através da propriedade Visible para que o código continuasse a ser executado e o formulário de mensagem foi exibido através de ShowModal, para aguardar que o usuário clique em um botão e só assim então prosseguir com a execução. O próximo passo seria criar o efeito contrário, neste caso o fade-out. Para tanto, o mesmo código do laço for deverá ser utilizado, porém ao invés de adicionar o valor de AlphaBlendValue, basta subtrair. Para concluir, o resultado da função será o valor de ModalResult do formulário da mensagem que foi definido quando o usuário clicou em um dos botões, e após seu retorno basta liberar os formulários da memória através de FreeAndNil().

Nota do DevMan

Quando se instancia um objeto, essa instância é referenciada por uma variável que aponta para o objeto em questão. Assim, ao se liberar um objeto da memória é muito importante também liberar a variável referenciada. Isso pode ser feito como é mostrado a seguir:

```
MeuObjeto := TMeuObjeto.Create;

MeuObjeto.Free;

MeuObjeto := nil;
```

Se não setarmos a variável utilizada para nil, é como se ela ainda fosse válida, porém, apontando para um objeto que não existe mais, o que pode causar falhas de memória (memory leaks). Para auxiliar o desenvolvedor nessa tarefa de liberação de memória, o Delphi introduziu o comando FreeAndNil, que substitui as duas linhas mostradas em uma só:

```
FreeAndNil(MeuObjeto);
```

É muito importante prestar atenção para situações como essa, pois são potenciais fontes de erro e podem ser muito difíceis de identificar.

Criando um Skin para um TButton

Atualmente existem muitos componentes que permitem que os desenvolvedores apliquem Skins, ou seja, altere toda a parte visual do programa, fugindo do tradicional fundo cinza e daquela barra de títulos repetitiva. Entretanto, tais componentes geralmente são pagos e consequentemente possuem seu código fonte fechado,

não permitindo que o usuário altere ou mesmo estude seu código. Neste exemplo será descrito como alterar o visual de um botão colocando em seu fundo uma imagem quando o mesmo não estiver com foco, outra para quando ele receber o foco do mouse e outra quando ele for clicado.

Para iniciar o desenvolvimento, a classe TButton será estendida, bem como alguns dos seus procedimentos. Para tanto, basta criar um tipo com o próprio nome da classe, ou seja, com o nome de TButton sendo igual a própria classe que contém o botão, em outras palavras, igual à StdCtrls.TButton. O código para estender a classe TButton deve ser inserido antes da criação do Formulário e seus objetos. O próximo passo é criar um objeto do tipo TCanvas para poder redesenhar o botão, uma vez que o mesmo não possui tal objeto. A classe TCanvas possui diversos controles gráficos que permitem desenhar textos, figuras, linhas, entre outros, consequentemente oferecendo ao desenvolvedor a opção de utilizá-la para redesenhar os objetos de acordo com sua preferência, uma vez que os controles já possuem um estilo visual conforme o tema do Sistema Operacional. Também será necessário manipular algumas mensagens do Windows (Windows Messages), que são informações enviadas do mesmo para a aplicação sobre determinada ação realizada, que não estão presentes nos eventos que o Delphi disponibiliza para o botão. Por exemplo, a mensagem CM_MOUSEENTER enviada pelo Windows quando o mouse está sobre o componente. Posteriormente, duas variáveis do tipo Boolean deverão ser criadas, sendo elas ComMouse, que indicará se o mouse está ou não sobre o botão, e a variável ComFoco, que indicará se o botão está em foco. Também será criado um TBitmap para carregar as imagens que serão utilizadas no botão. O próximo passo é estender alguns dos procedimentos e criar outros para auxiliar. Para tanto, basta utilizar override, permitindo assim que um procedimento ou função que já existe seja sobreposto ou estendido. Neste caso, para estender o procedimento, basta utilizar inherited no mesmo, fazendo com o código anterior dele também seja executado, ou seja, fazendo com que o procedimento seja herdado da sua classe principal. A **Listagem 7** demonstra o que foi citado anteriormente.

O primeiro procedimento estendido é o construtor, ou seja, o método Create() do próprio botão que é executado quando o mesmo é criado. Este procedimento é estendido porque nele são criados um objeto do tipo TCanvas para que o mesmo seja redesenhado futuramente, e também um TBitmap.

Como foram criados no construtor dois novos objetos para o botão, no caso o Canvas e o Bitmap, no procedimento destruidor (Destroy) do mesmo também deverá ser estendido para que possa liberar da memória os objetos quando o botão for destruído. O código a seguir mostra como isso é feito:

```
destructor TButton.Destroy;
begin
  FreeAndNil(FCanvas);
  FreeAndNil(Bmp);
  inherited;
end;
```

Listagem 7. Métodos necessários para redesenhar o TButton com as devidas propriedades

```
type
  TButton = class(StdCtrls.TButton)
  private
    FCanvas: TCanvas;
    FBmp: TBitmap;
    ComFoco, ComMouse: Boolean;
  protected
    procedure CreateParams(var Params: TCreateParams); override;
    procedure WndProc(var Message : TMessage); override;
    procedure SetButtonStyle(Value: Boolean); override;
    procedure DesenhaBotao(Rect: TRect; State: UINT);
    procedure CNDrawItem(var Message: TWMDrawItem);
    message CN_DRAWITEM;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  end;
```

O próximo passo é estender o procedimento CreateParams(). Quando um objeto é derivado de um TWinControl, como um TForm, TButton, TEdit, entre outros, estender este procedimento é muito útil quando se deseja alterar algumas propriedades de tal objeto que são definidas por padrão pelo sistema operacional, como seu estilo visual por exemplo. Vale também destacar que o CreateParams() é executado no momento da criação do mesmo, sendo assim, para informar ao Windows que o estilo do botão será desenhado pela aplicação, basta alterar o parâmetro Style e defini-lo como "BS_OWNERDRAW", em outras palavras, que tal objeto será desenhado pelo próprio desenvolvedor. Portanto, outro procedimento deverá ser estendido futuramente para desenhar

o botão. No código a seguir está demonstrado como estender o método CreateParams() e defini-lo de acordo com o desejado:

```
procedure TButton.CreateParams(var Params: TCreateParams);
begin
 inherited;
 Params.Style := Params.Style or BS_OWNERDRAW;
end;
```

Depois de estendido o CreateParams(), o procedimento SetButtonStyle() deverá ser estendido para verificar e definir o valor da variável criada anteriormente, ComFoco, para saber se o botão recebeu ou não o foco. Também será necessário incluir o procedimento Invalidate da Unit Controls para informar ao sistema que o atual controle deve ser redesenhado caso o valor de ComFoco seja alterado. A seguir, como fica todo esse procedimento:

```
procedure TButton.SetButtonStyle(Value: Boolean);
begin
 if Value <> ComFoco then
 begin
  ComFoco := Value;
  Invalidate;
 end;
end;
```

Também é necessário estender o procedimento WndProc() do botão. O método WndProc() permite processar mensagens do Windows, tornando possível obter informações sobre o mouse, por exemplo. Ou seja, neste caso se o mouse está sobre o botão ou não para criar o efeito de MouseOver, e MouseOut, quando perder o foco. Esse controle fica na variável criada anteriormente, ComMouse. Evidentemente, será necessário utilizar o método Invalidate para redesenhar o componente novamente. Na **Listagem 8** se pode visualizar como efetuar tal procedimento e como obter a informação através do valor de Message.Msg, comparando se igual a "CM_MOUSELEAVE", perdendo o foco do mouse, ou "CM_MOUSEENTER", recebendo o foco do mouse.

Depois de criado o conjunto de procedimentos utilizados para manipular as informações, será criado o procedimento que vai desenhar ou redesenhar o botão. Tal procedimento recebeu o nome de DesenhaBotao(). Na verdade, ele verificará as informações obtidas pelos outros procedimentos, como se o mouse está sobre o componente através da variável ComMouse, e de acordo com seu valor, utilizará o objeto Canvas criado anteriormente no método Create() para desenhar a imagem correspondente bem como a propriedade Caption do botão. Vale a pena destacar que no momento em que foi informado ao Windows que o controle seria desenhado pela aplicação, tudo que faz parte do visual do mesmo deverá também ser redesenhado, ou seja, se foi informado um rótulo para o botão, o mesmo não será escrito automaticamente, devendo então ser escrito manualmente através do Canvas.

O procedimento DesenhaBotao() deverá ter um parâmetro do tipo TRect, que permite guardar as coordenadas de um objeto retangular, para armazenar o tamanho do botão que está sendo processado atualmente, uma vez que este procedimento será reutilizado várias vezes para redesenhar todos os botões do atual formulário, levando em consideração que cada botão pode ter um tamanho diferente. Também deverá ter um parâmetro do tipo "Uint", LongWord, para informar o estado de cada botão. Serão criadas algumas variáveis para obter outros estados (State) do botão que não foram citadas anteriormente, como se o mesmo está desabilitado ou pressionado. Posteriormente, as Flags serão alteradas de acordo com os valores das variáveis. Vale ressaltar que através de Flags, neste caso, é possível informar quais opções ou propriedades estão ativas no botão. A **Listagem 9** mostra o método DesenhaBotao até este momento.

Agora é necessário verificar o valor presente em cada variável para desenhar a imagem adequada no botão. No exemplo proposto no artigo, o objeto TBitmap criado anteriormente será utilizado para carregar a imagem adequada direto da pasta onde está a aplicação, permitindo assim a substituição da imagem sem ter que alterar o executável. O desenvolvedor pode utilizar de outros meios para tanto, se preferir, como criar uma biblioteca dentro do executável contendo todas as imagens evitando assim que elas sejam carregadas do disco. O interessante é utilizar o recurso StretchDraw() do objeto Canvas para fazer com a imagem seja ajustada ao tamanho do botão. Para dar sequência, verifica-se se o valor de ComFoco ou Padrao é verdadeiro, e carrega-se a imagem padrão para o botão. Verifica-se se o valor de ComFoco é falso e carrega-se a imagem quando o botão está sem foco. Verifica-se se o valor da variável Pressionado é verdadeiro e carrega-se a imagem quando o botão for pressionado, bem como se altera a posição do texto para criar um efeito de profundidade.

Listagem 8. Resgatando status de posicionamento do mouse

```
procedure TButton.WndProc(var Message :TMessage);
begin
 if (Message.Msg = CM_MOUSELEAVE) then
 begin
  ComMouse := False;
  invalidate;
 end;
 if (Message.Msg = CM_MOUSEENTER) then
 begin
  ComMouse := True;
  invalidate;
 end;
 inherited;
end;
```

Listagem 9. Partes do método desenha botão

```
procedure TButton.DesenhaBotao(Rect:TRect; State: UINT);
var
 Flags: Longint;
 Pressionado, Padrao, Desabilitado: Boolean;
begin
 Flags := DFCS_BUTTONPUSH or DFCS_ADJUSTRECT;
 Pressionado := State and ODS_SELECTED <> 0;
 Desabilitado := State and ODS_DISABLED <> 0;
 Padrao := State and ODS_FOCUS <> 0;
 if Pressionado then Flags := Flags or DFCS_PUSHED;
 if Desabilitado then Flags := Flags or DFCS_INACTIVE;
...
```

Verifica-se também se o valor de ComMouse é verdadeiro e Pressionado é falso para criar o efeito de MouseOver descrito anteriormente. Para concluir, se o valor de Desabilitado estiver verdadeiro, então desenha-se o efeito relativo a um componente desativado do Windows, para tanto basta utilizar o Objeto Canvas e posteriormente DrawState(), passando os parâmetros necessários. Logicamente se Desabilitado for falso, ou seja, se o componente está habilitado, simplesmente pode-se mandar desenhar o texto do Caption do botão utilizando seu objeto Canvas através de DrawText(). Vale a pena ressaltar também que a fonte, bem como seu estilo, pode ser alterada pelo objeto Canvas no momento em que o botão for redesenhado. A **Listagem 10** demonstra parte do procedimento que faz o que foi descrito.

Para finalizar a aplicação de Skin no TButton é criar o procedimento CNDrawItem() que envia a mensagem CN_DRAWITEM ao Windows para desenhar o botão. Também será necessário utilizar o método DrawItemStruct da API do Windows. Ele permite obter informações dos controles, como por exemplo, o parâmetro itemState, assim o desenvolvedor consegue determinar se tal controle deve ser desenhado em seu estilo padrão, com foco, entre outros. Também consegue obter o handle do dispositivo de contexto. A **Listagem 11** mostra como codificar o procedimento CNDrawItem().

Conclusão

O uso de efeitos gráficos em aplicações desktop tem se tornado cada vez mais comum graças ao surgimento e aperfeiçoamento de tecnologias e ferramentas que permitem que tais efeitos sejam desenvolvidos de uma maneira mais rápida e menos

Listagem 10. Visualizando os parâmetros dos métodos do canvas

```
...
 FCanvas.Brush.Style := bsClear;
 FCanvas.Font.Color  := clWhite;
 FCanvas.Font.Style := [fsBold];
 if Desabilitado then
  DrawState(FCanvas.Handle, FCanvas.Brush.Handle, nil, Integer(Caption), 0,
  ((Rect.Right - Rect.Left) - FCanvas.TextWidth(Caption)) div 2,
  ((Rect.Bottom - Rect.Top) - FCanvas.TextHeight(Caption)) div 2,
   0, 0, DST_TEXT or DSS_DISABLED)
 else
  DrawText(FCanvas.Handle, PChar(Caption), - 1, Rect,
   DT_SINGLELINE or DT_CENTER or DT_VCENTER);
end;
```

Listagem 11. Codificando o evento CNDrawItem

```
procedure TButton.CNDrawItem(var Message: TWMDrawItem);
var
 SalvarIndice: Integer;
begin
 with Message.DrawItemStruct^ do
 begin
  SalvarIndice := SaveDC(hDC);
  FCanvas.Lock;
  try
   FCanvas.Handle := hDC;
   FCanvas.Font := Font;
   FCanvas.Brush := Brush;
   DesenhaBotao(rcItem, itemState);
  finally
   FCanvas.Handle := 0;
   FCanvas.Unlock;
   RestoreDC(hDC, SalvarIndice);
  end;
 end;
 Message.Result := 1;
end;
```

Faça um *upgrade* em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre ouutros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software. Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviços); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**

DevMedia

trabalhosa, sem contar que muitas vezes chama a atenção do usuário final. Também é possível observar que o ambiente do Delphi, utilizando a VCL, permite criar vários efeitos sem que o desenvolvedor necessite optar por outra ferramenta, todavia como todo desenvolvimento gráfico, exige um pouco mais de tempo, dedicação e certo conhecimento sobre o funcionamento dos objetos no ambiente Windows. É sempre importante lembrar que o desenvolvedor pode criar seus próprios componentes de efeitos para utilizar na aplicação toda e em futuros projetos. Como foi observado, manipulando as mensagens do Windows, eventos e as propriedades da aplicação, o desenvolvedor pode criar skins e aplicá-los em vários componentes, deixando a aplicação com um visual diferenciado, rico e agradando principalmente o usuário final. É interessante lembrar também, que enquanto não se pode utilizar os recursos do novo FireMonkey, já que são incompatíveis com a VCL, não pode-se deixar de atualizar os sistemas legados, utilizando desses artifícios simples para agradar o usuário final. Também é possível aplicar efeitos gráficos no Delphi utilizando a GLScene. Ela é uma biblioteca de recursos 3D baseada em OpenGL que oferece vários componentes visuais e objetos que permitem a realização de cenas 3D. Vale destacar também que ela vem crescendo com o decorrer do tempo, ganhando vários recursos para um motor 3D genérico (3D Engine), sem contar que é open-source e oferece suporte a áudio. Também permite criar tais cenas e animações 3D sem a necessidade do domínio dos procedimentos e funções da OpenGL. A GLScene também é encontrada como um componente que pode ser adicionado ao Delphi, inclusive traz uma diversidade de exemplos que são muito úteis para uma familiarização com a biblioteca. Portanto, o desenvolver que desejar criar efeitos gráficos mais poderosos para a aplicação, poderá se utilizar deste componente.

Jederson Zuchi
jederson@sygnux.com.br
É diretor de sistemas da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP. Formado pela Fatec/TQ, desenvolve com Delphi desde a versão 3. Atualmente trabalhando com Delphi, Java e Plataforma .NET em diversos ERPs industriais. Professor da Fatec/TQ lecionando em diversas matérias de desenvolvimento de software. Twitter @jedersonzuchi.

Giuliano Scombatti Pinto
proctec@hotmail.com
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java e PHP. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Tutoriais e Download da GLScene
http://glscene.sourceforge.net/wikka/HomePage

Tutoriais sobre OpenGL
http://www.opengl.org/code/

Blog do autor
http://twitter.com/jedersonzuchi

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

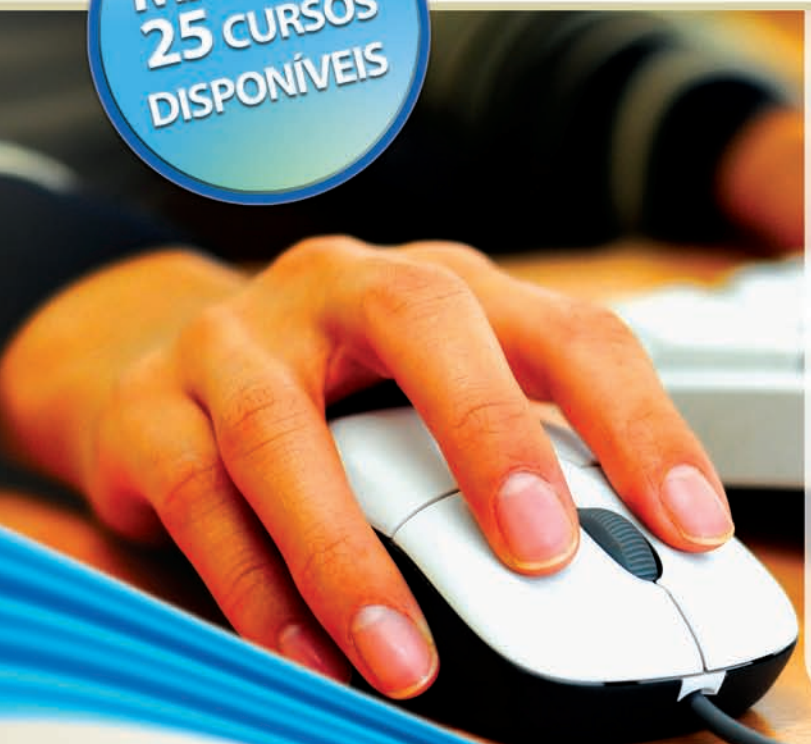
www.devmedia.com.br/clubedelphi/feedback

CURSOS ONLINE

.net magazine

A Revista .net Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

MAIS de 25 CURSOS DISPONÍVEIS



CONHEÇA OS CURSOS MAIS RECENTES:

- **Curso .net Entity Framework - Loja Virtual**
- **Novidades do Visual Studio 2010**
- **Componentes DevExpress**
- **Crie um chat utilizando Asp.Net 3.5 e Linq**
- **Novidades do C# (csharp) 3.0**
- **POO com C#**
- **Microsoft Visual Studio Tools for Office (VSTO)**
- **Computação na Nuvem com a Plataforma Microsoft Windows Azure**

Para mais informações :

www.devmedia.com.br/curso/netmagazine

(21) 3382-5038

http://www.devmedia.com.br

DevMedia

Firebird: Poderoso, leve,

Mini Curso – Parte 1

Resumo DevMan

De que trata o artigo:

O artigo mostrará o Firebird, começando desde seu início, através da liberação do código-fonte da Interbase 6.0 pela Borland, mostrando os recursos que este poderoso e leve banco de dados possui e o porquê de usar o Firebird ao invés de outros bancos de dados não comerciais.

Em que situação o tema é útil:

Infelizmente, muitas pessoas têm uma visão errônea do Firebird! Muitos acham que o Firebird não possui recursos de banco de dados, como controle de integridade referencial, triggers, stored procedures, tabelas derivadas; outros acham que ele serve apenas para programas e clientes pequenos e que o banco de dados não suporta tantas transações; há ainda aqueles que acreditam que o Firebird só pode ser usado para guardar dados, como se fosse arquivos DBF; outros que acham que o Firebird só consegue armazenar dados de poucos gigabytes. A ideia do artigo é mostrar que estes mitos são errôneos e sem fundamentos, mostrando a evolução do Firebird através dos anos e mostrar os principais recursos deste banco de dados.

Firebird: Poderoso, leve, gratuito e Open-Source:

Este artigo mostrará como o Firebird pode ajudar no dia a dia, através de recursos que o banco traz e quebrar muitos preconceitos que formaram deste banco de dados. O leitor conseguirá ver que o Firebird é tão poderoso como os demais bancos de dados do mercado, como Oracle, DB2, SQL Server, PostgreSQL, MySQL. Você leitor, verá e sentirá confiança em trabalhar com um banco de dados total free e open-source, onde a premissa do mesmo é não precisar de um DBA. Será uma série de artigos, começando com a história e um breve resumo dos principais recursos e funções. Ao final da série de artigos, espero que você leitor, possa ter a plena confiança em usar este banco e poder fazer parte desta enorme comunidade, que é presente em todo mundo.

Este primeiro artigo abordará a história do Firebird, um resumo dos principais recursos e veremos as diferenças das versões do servidor. A ideia é quebrar pré-conceitos existentes em torno do Firebird e mostrar que este banco de dados é capaz de gerenciar os dados em qualquer situação, como sistemas de missão crítica (24x7 por exemplo), sistemas com múltiplas conexões simultâneas, entre outras e iremos aprender como isso pode nos beneficiar, sabendo utilizar cada recurso.

Possui uma comunidade bem ativa, onde todos cooperam com a criação e tradução de documentos oficiais, além no desenvolvimento, testes e melhorias deste sistema gerenciador de banco de dados.

E o melhor de tudo, sem custo para aquisição de licenças, sem custo para suporte e sem custos para atualizações de versões, pois o Firebird é free e possui o código-fonte open source. É um banco de dados leve, mas poderoso e que não necessita de um profissional DBA para ficar monitorando o banco de dados.

História

O Firebird é descendente "direto" (filho) do Interbase 6.0 da Borland. A Borland decidiu abrir o código-fonte em 25 de julho de 2000 sob a licença IPL,

gratuito e Open-Source

Fabiano Moura

Interbase Public License. O lançamento do Interbase 6.0 como fonte aberto, foi muito bem recebido pela comunidade, pois o Interbase era reconhecido mundialmente como sendo um banco de dados eficiente e que possuía uma simplicidade na instalação e manutenção do banco de dados.

Através da iniciativa de alguns ex-funcionários da Borland que tinham feito parte do desenvolvimento do Interbase, decidiram criar uma árvore dos fontes no SourceForge, em paralelo, nascendo assim o Firebird.

A partir daí, muitos usuários que apenas conhecia o Interbase, começaram a estudar o código- fonte, corrigindo diversos erros e fazendo diversas melhorias no fonte. Chegaram encontrar uma falha de segurança grave, backdoor, aonde qualquer pessoa que conhecia o usuário e senha do backdoor, tinha acesso total a máquina aonde estava instalado o Interbase.

Após algum tempo do código-fonte ser liberado pela Borland, a mesma anunciou que continuaria a vender o Interbase com algumas melhorias e recursos a mais que o código-fonte aberto tinha. Através desta medida, a diferença entre os fontes do Interbase e Firebird começou a acontecer e a dificuldade entre troca de informações entre a equipe do Firebird e a equipe do Interbase (Borland). O Firebird 1.0, trouxe várias melhorias que a comunidade do Interbase pedia a anos, como: implementação de páginas de 16 K; porta padrão passou a ser 3050; descriptors em UDFs (permitia passar e receber valor NULL das UDFs); UDFs e Generators monitorados; suporte a arquivos maiores; Group by UDF; Recreate Procedure/ Table; maior suporte ao ANSI SQL; entre outros. Mesmo assim, ambos ainda eram compatíveis até a versão 1.5 do Firebird. Somente a partir da versão 2.0 que, Firebird e Interbase tornaram-se totalmente incompatíveis.

Fundação FirebirdSQL

A fundação FirebirdSQL nasceu com o intuito de levantar fundos no mundo inteiro para conceder a desenvolvedores que estavam trabalhando no código-fonte do Firebird, criando novos recursos, testando e corrigindo o Firebird. A fundação é sem fins lucrativos e está estabelecida em New South Wales, Austrália.

Como forma de representar o Firebird, a comunidade juntamente com a fundação, escolheu a Fênix Sparky, como mascote e símbolo do Firebird. A primeira aparição, ocorreu em Fulda, Alemanha em maio de 2003.

Recursos

O Firebird é um software Cliente/Servidor, projetado inicialmente para uso local e em redes WAN, por isso, o núcleo do servidor Firebird, possui dois programas: o servidor de banco de dados, que é executado e hospedado em um computador de rede e a biblioteca cliente, que faz a comunicação com o servidor.

Os binários do Firebird 1.0 foram desenvolvidos e corrigidos, melhorando vários módulos na linguagem C. Já a partir do Firebird 1.5, os módulos foram reescritos em C++. Apesar da mudança da linguagem, a interface de programação de aplicações (API) não foi alterada, ou seja, softwares escritos para o Firebird 1.0 requerem pouca ou nenhuma mudança para funcionar com o Firebird 1.5. A seguir, iremos ver com mais detalhes os recursos e arquiteturas do Firebird.

Suporte a sistemas operacionais

Ao contrário do Interbase 6 que possuía suporte apenas para Win32, Linux e Solares, o Firebird, desde a versão 1.0, possui suporte Win32, Linux, Solaris, Mac OS X, FreeBSD, NetBSD, AIX, HP-UX e SCO, ou seja, o Firebird pode ser executado em qualquer plataforma que aceita conexões TCP/IP e toda plataforma cliente que possa implementar a API do Firebird.

Acesso a rede

Apesar de o Firebird poder ser executado em qualquer sistema operacional que implemente o TCP/IP e sua API, o mesmo não consegue se conectar através de qualquer meio de compartilhamento de sistemas de arquivos, ou seja, compartilhamentos NFS, Samba, unidades mapeadas do Windows, entre outros, não podem ser usados como locais de armazenamento do banco de dados. O usuário deverá sempre armazenar em dispositivos locais, como uma partição qualquer do HD. Outro detalhe muito importante é o fato do cliente Firebird conseguir se conectar a um caminho físico absoluto, ou seja, é necessário informar o IP/Porta:caminho_do_banco_de_dados. Porém, a partir do Firebird 1.5, foi adicionado o recurso de alias de banco de dados, onde o caminho do banco ficará no arquivo aliases.conf e o cliente Firebird não necessitará de um caminho absoluto, pois o mesmo irá aplicar o caminho do alias do banco de dados.

"Multigeracional"

O Firebird possui a capacidade de isolar e controlar múltiplas gerações de um único registro simultaneamente, evitando problemas de deadlock. Isso acontece cada vez que um usuário tenta acessar o mesmo registro simultaneamente, o servidor Firebird faz o controle destas versões e só considera a versão mais recente confirmada. Esta versão será a vista fora da tarefa do usuário que enviou com sucesso uma nova versão. O(s) usuário(s) só ficará(ão)

ciente(s) que aconteceu algo com o registro após o mesmo ser bloqueado contra atualização ou exclusão do mesmo registro, até que a nova versão se torne oficial.

Integridade Referencial

O Firebird possui suporte a integridade referencial, conforme o padrão SQL, ou seja, possui suporte a atualizações, exclusões em cascata, com uma diversidade grande de opções de gatilhos (triggers).

Sombreamento de banco de dados

O servidor Firebird possui a possibilidade de manter uma cópia em tempo real do banco de dados com alguns atributos extras, que a tornam indisponível para leitura. É importante ficar atento que isso não é uma replicação de banco de dados. O sombreamento acontece no mesmo servidor, ou seja, no mesmo hardware que está sendo executado o servidor de banco de dados.

Views (visões)

Views são objetos do banco de dados, semelhante a uma tabela, podendo ser usado para leitura através de uma recuperação simples dos dados de uma tabela ou de N tabelas através de Joins, por exemplo. Caso a view seja criado através de uma recuperação de dados de uma única tabela, há a possibilidade de fazer insert, update e delete através das views.

Triggers (gatilhos)

Triggers são pequenos programas armazenados no banco de dados, que possui uma função específica e momento específico de execução. O trigger no Firebird pode ter vários meios de ações, ou seja, Insert, Update e Delete e pode ser escolhido o tempo, se será após uma ação, ou antes, de uma ação. Através de triggers, podemos fazer qualquer coisa no banco de dados, como por exemplo, controle do estoque, integração com o contas a pagar e receber, integração com o faturamento, integração com a contabilidade. A linguagem usada pelo Firebird para criação deste programa é o PSQL.

Stored Procedures

Também são pequenos programas, porém, com a diferença que não são executados automaticamente após uma ação. Esses programas são armazenados no banco de dados e podem ser chamados pelo programa cliente, para ser feito alguma rotina dentro do banco de dados, como por exemplo, atualização de um campo, como ser usado como uma tabela, ou seja, podendo retornar dados para o programa cliente.

Transações

O Firebird, como mencionado acima, possui a possibilidade de gerenciar múltimas transações com o banco de dados e com o mesmo registro, não bloqueando o usuário que está lendo o registro. Isso torna o Firebird muito produtivo, pois ao ler um único registro por mais de um usuário não é travado (Deadlock), impossibilitando a leitura do mesmo. A trava somente acontece, se mais de um usuário tentar modificar ou excluir o mesmo registro.

Segurança

O Firebird fornece a possibilidade de acesso ao banco de dados através de usuário e senhas criptografadas, mas como qualquer servidor de banco de dados, o mesmo baseia-se na segurança adequada da rede e do sistema de arquivos. O Firebird possui a possibilidade de armazenar dados criptografados, mas não fornece ainda o suporte de criptografar os próprios dados.

Privilégios SQL

Exceto o usuário SYSDBA e o dono do banco de dados, nenhum outro usuário possui acesso a qualquer objeto e aos dados armazenados no banco de dados, a não ser que os mesmos tenham dito acesso para isso.

Campos Calculados

São campos que os seus dados, não são armazenados no banco de dados. Os valores destes campos são criados no momento que o dado é recuperado através de um select, por exemplo, e são criados através de funções ou expressões, tanto internas do Firebird, quanto de UDFs.

ANSI SQL, Generators, Charsets e Collations

Suporta quase todo o padrão ANSI SQL 92.

Nota do DevMan

O ANSI SQL é um forum que existe para discutir a padronização da linguagem SQL, independentemente de fabricante ou versão específica de gerenciador de banco de dados. Sempre que o padrão é atualizado, sua identificação é feita pelo ano dessa atualização. Por exemplo, a versão Ansi SQL 92 foi definida em 1992. A versão mais recente é a 2008.

Generators são objetos do banco de dados, aonde recebem valores únicos do núcleo do Firebird, não dando a possibilidade de ser repetidos. É usado bastante para determinar o valor da próxima chave primária.

Charset define a forma como o servidor de banco de dados irá traduzir um carácter. O charset também é responsável por determinar quais tipos de caracteres serão aceitos. Cada charset há um colletions associado.

Collations são estruturas que definem o comportamento do servidor Firebird quanto ao tratamento de caracteres no momento de uma ordenação ou comparação entre eles, por exemplo, se José ou Jose devem ser ordenados da mesma forma.

Índices

São objetos de banco de dados, mais precisamente, ponteiros lógicos, aonde servirá para a localização física das linhas de uma tabela. Na maioria das vezes o servidor de banco de dados, examina os índices para localizar um registro no banco de dados. O uso de índices geralmente traz rapidez na recuperação (consulta) de registros de banco de dados. A partir do Firebird 1.5 em diante, o limite de quantidade de índices por tabela é de 256.

Alguns índices são criados automaticamente para manter uma integridade referencial, ou seja, são índices de chaves primárias e chaves estrangeiras.

Domínios

São tipos de dados definidos pelo usuário, que tem o objetivo de padronizar e agilizar manutenções em tabelas, stored procedures e triggers que utilizem domínios. As colunas que são definidas por domínios herdam todos os tipos de atributos tais como: tipo de dado; valor default; status NULL ou Not NULL; restrições; conjunto de caracteres; prdem de intercalação.

Blobs e Matrizes

Blobs são tipos de dados complexos, usados para armazenar objetos de dados discretos de tamanho variável. São considerados complexos por sua forma de armazenamento interno e não quanto ao seu uso. No tipo Blob pode ser armazenado qualquer tipo de dados, como imagem, vetor, vídeos, documentos textos, etc. Alguns detalhes devem ser levados em consideração. Do lado servidor, os campos blobs não possuem índices. Do lado cliente, o mesmo precisa ser tratado de forma correta e necessária para poder recuperar o dado armazenado de forma correta. O tipo de dado Array possibilita armazenar dados de forma multidimensional em uma única coluna. O servidor Firebird, consegue realizar operações sobre um array inteiro ou sobre um pedaço de array.

Usuários

O Firebird possui o usuário administrador chamado SYSDBA. Este usuário é super poderoso e não tem limites. É aconselhável sempre trocar a senha deste usuário e sempre utilizar outro usuário para criar e dar manutenção ao banco de dados. Atualmente até o Firebird 2.5, os usuários não são guardados no banco de dados, ou seja, mesmo mudando a senha do SYSDBA, caso alguém tiver acesso físico ao banco de dados, conseguirá copiar e acessar os dados através de outra instalação do servidor Firebird. Este recurso de segurança, deverá ser implementado no Firebird 3.0.

Grupos

O Firebird tem a possibilidade da criação e controle de privilégios a SQL para usuário. Automaticamente, ao definir que um usuário faz parte deste grupo, o mesmo herda todos os privilégios que este usuário possui. Isto é muito importante, quando tem a necessidade de dar o mesmo privilégio a vários usuários do banco de dados.

UDFs (User Defined Functions)

O Firebird possui um número de funções muito vasto, tanto de padrões SQL, quanto próprios, mas há a possibilidade de criar funções em qualquer linguagem que gere bibliotecas compartilhadas, no windows chamado de DLL, para serem usados no servidor de banco de dados.

Execute Block

São programas parecidos com as stored procedures, mas que não são armazenados no banco de dados.

Possibilidade de rodar mais de um servidor Firebird

Há a possibilidade de executar mais de um servidor Firebird no mesmo servidor. Esse procedimento não é comum, mas as vezes é necessário, principalmente quando temos programas que rodam apenas em determinadas versões do Firebird. Como é o caso dos programas do Ministério da Saúde (DataSUS), onde a grande maioria dos programas tem suporte apenas ao Firebird 1.5. Para que isso possa ser feito não podemos utilizar o wizard da instalação e sim o servidor Firebird compactado e que pode ser baixado no site da SourceForge. Deveremos também mudar a porta de comunicação, ou seja, cada versão do servidor deverá trabalhar com uma porta de comunicação diferente.

Auditoria e tabelas de monitoramento

A partir do Firebird 2.0 foram incluídas tabelas de monitoramento, da qual podemos ver em tempo real o que está acontecendo com o banco. Podemos ter ideia de quais comandos (querys) foram executados, quanto tempo levou para executar, qual o usuário que executou, qual o aplicativo, entre outros. No Firebird 2.5 foi incluído um plugin chamado Trace API que faz a auditoria do banco de dados, mostrando para nós tudo o que acontece com o mesmo. Através desta auditoria e da tabela de monitoramento, podemos analisar os dados do que está acontecendo com o banco de dados Firebird.

Nota do DevMan

Replicação de banco de dados refere-se a um backup do banco de dados em locais distintos, podendo ou não ser online. Este backup pode ser de via única, ou seja, do servidor de produção para um servidor de backup, ou de várias vias, como do servidor de produção máster, para outros servidores de produção e o inverso também.

A replicação também pode ser usada como forma de balanço de dados, evitando uma sobrecarga do servidor e podendo agilizar a comunicação entre matriz e filial, por exemplo.

Ferramentas de Administração

O Firebird possui várias ferramentas para administração do banco de dados. Veremos algumas delas e o que elas fazem e como podem nos ajudar no dia a dia.

Na pasta de instalação do Firebird, precisamente na basta Bin (**Figura 1**), vemos vários arquivos, alguns deles podem variar dependendo do tipo de servidor instalado.

Fbclient.dll

O arquivo fbclient.dll é o cliente do Firebird. É ele que será responsável por fazer a comunicação da nossa aplicação, com o servidor Firebird.

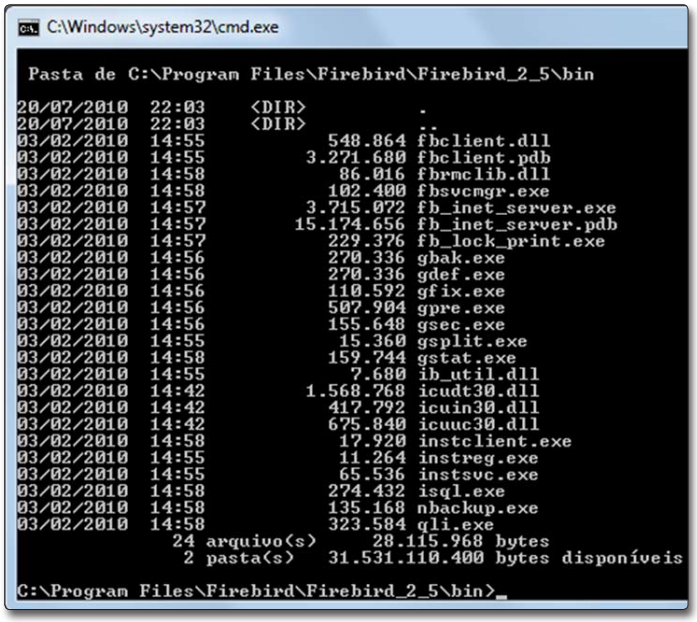


Figura 1. Listagem dos arquivos na pasta Bin da instalação do Firebird

Instsvc.exe

Este aplicativo tem a função de instalar o servidor Firebird como serviço nas versões do Windows que suportam esse recurso. O comando mais básico que podemos usar para utilizar este recurso é instsvc install. Ao digitar este comando, o aplicativo fará que o Firebird seja instalado como serviço. O instsvc fornece vários parâmetros para instalação. Vejamos quais a seguir:

```
instsvc i[nstall] [-s[uperserver]* | -c[lassic]]
 [-a[uto] * | -d[emand]]
 [-g[uardian]]
 [-l[ogin] username [password]]
 [-i[nteractive]]
sta[rt] [-b[oostpriority]]
sto[p]
q[uery]
r[emove]
```

Caso queira ver se o serviço iniciou de forma correta, digite o comando: instsvc query . O mesmo deverá retornar o status do serviço.

Instreg

Este aplicativo cria no registro do windows, a chave padrão (HKLM\SOFTWARE\Fifrebird Project\Firebird Server\Instances) que indica o diretório onde o servidor Firebird está instalado. Os parâmetros são:

```
instreg i[nstall]
 r[emove]
 [-z]
```

O -z mostra a versão atual do servidor Firebird.

Instclient

Esse aplicativo é responsável pela instalação da biblioteca cliente do Firebird, fbclient.dll, ou a biblioteca de retrocompatibilidade com o Interbase, gds32.dll, na pasta de sistema do windows. Os parâmetros são demonstrados a seguir:

```
instclient i[nstall] [-f[orce]] library
 q[uery] library
 r[emove] library
```

Sendo que library é f[bclient] ou g[ds32].

A biblioteca gds32.dll no Firebird existe apenas para manter a compatibilidade existente com programas que a requerem, seja porque foram feitos para alguma versão mais antiga do Firebird ou porque esperam um servidor Interbase.

Isql

A principal função de aplicativo é fornecer um ambiente aonde o usuário poderá conectar-se ao banco de dados e fazer as devidas manutenções, como create, alter, drop, além das operações de CRUD como insert, update, delete. O utilitário isql possui outras funções também. Digite isql -help e veja a lista de parâmetros aceitos pel isql (**Figura 2**).

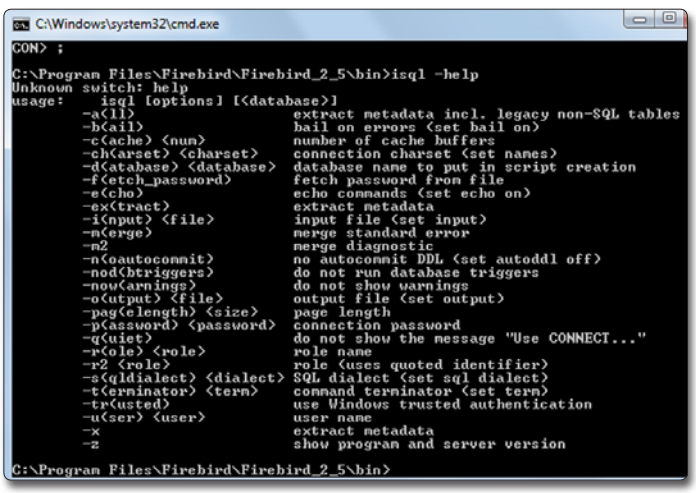


Figura 2. Parâmetros do aplicativo isql

Para que possamos conectar a algum banco de dados, utilizamos comando a seguir:

```
isql -u SYSDBA -p masterkey "c:\programs file\firebird\firebird_2_5\examples\empbuild\
employee.fdb"
```

Para sair do isql existem dois comandos: o quit, que faz rollback das transações ainda abertas e o exit, que faz o commit para as transações em aberto.

Uma outra funcionalidade muito importante, ao meu ver, é poder extrair todo o metadata dos objetos do banco de dados. Para isso, basta executarmos o comando a seguir:

```
isql -extract -o "c:\temp\metadata.sql" "c:\programs file\firebird\firebird_2_5\examples\
empbuild\employee.fdb" -u SYSDBA -p masterkey;
```

Uma observação sobre o isql. Todo comando termina com o terminador ponto-vírgula (;).

Gsec

É o aplicativo responsável pela criação e troca da senha do usuário. Para sabermos as lista de parâmetros completos, digitamos gsec -help. Veja a **Figura 3**.

Para criar um usuário, basta digitar o comando abaixo:

```
gsec -user SYSDBA -password masterkey
```

Em seguida, digite o comando:

```
add CLUBEDELPHI -pw delphi
```

Como o Firebird guarda as senhas no formato binário e caso você crie um usuário com uma senha maior de sete caracteres, será ignorado o restante dos caracteres após o sétimo caracter.

Para sair do gsec, digite quit.

Gfix

Este aplicativo é utilizado para realizar uma validação ou até mesmo correção estrutural de um banco de dados. Este checagem serve para verificarmos se o banco está ou não corrompido internamente. Para fazermos esta checagem, basta digitar o comando a seguir:

```
gfix -v -full -user SYSDBA -password masterkey c:\caminho_do_banco\bancodedados.fdb
```

Caso retorne erro, podemos fazer a preparação do arquivo de banco de dados, falando para o banco de dados ignorar os dados corrompidos. Para que possamos fazer isso, digitamos o comando a seguir:

```
gfix -mend -full -user SYSDBA -password masterkey c:\caminho_do_banco\
bancodedados.fdb.
```

A possibilidade do uso do gfix é grande. Para saber todos os parâmetros que o mesmo suporta, basta digitar o comando gfix. Veja a **Figura 4**.

Gbak

Com este aplicativo conseguimos realizar o backup do banco de dados com ele ainda em utilização. É importante nunca copiar o arquivo do banco de dados com ele em uso, pois poderá danificar o arquivo. Para podermos fazer o backup com ele em uso, utilizamos o aplicativo gbak.

Veja a **Figura 5**, os parâmetros possíveis de entrada.

Resumindo, para se fazer um backup gerando o arquivo de backup aonde estivermos executando o aplicativo, utilizaremos a linha de comando a seguir:

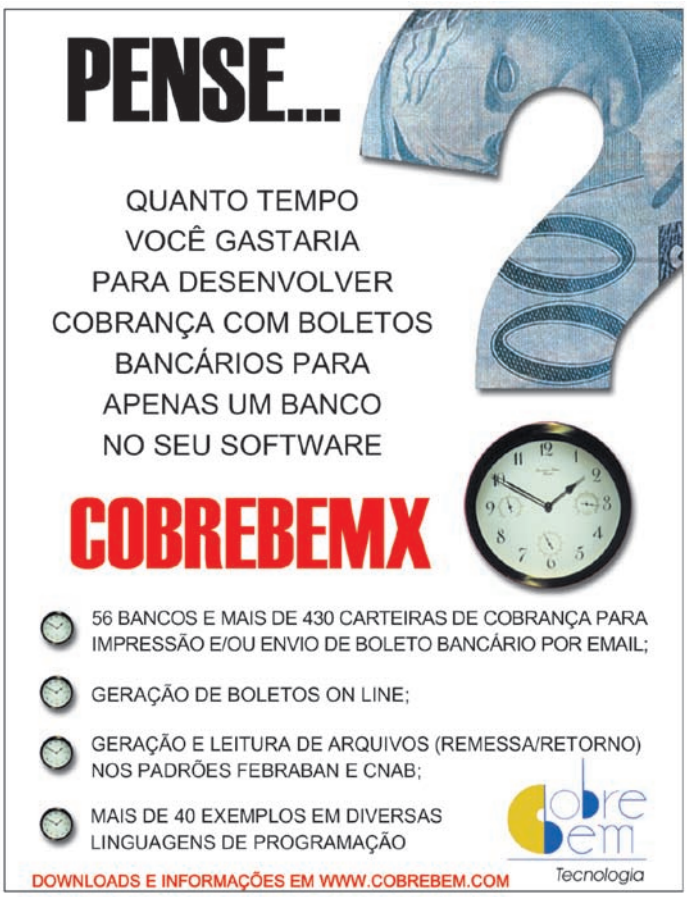
```
gbak -b -t -v -user SYSDBA -password masterkey IP/porta:c:\caminho_do_banco_no_
servidor\bancodedados.fdb c:\caminho_do_backup\nomebackup.fbk
```

Para restaurar, utilizamos o comando a seguir:

```
gbak -C -user SYSDBA -password masterkey c:\caminho_do_backup\nomebackup.
fbk c:\caminho_do_banco\banco.fdb
```

```
gsec utility - maintains user password database

command line usage:
  gsec [ <options> ... ] <command> [ <parameter> ... ]

interactive usage:
  gsec [ <options> ... ]
  GSEC>
  <command> [ <parameter> ... ]

available options:
  -user <database administrator name>
  -password <database administrator password>
  -fetch_password <file to fetch password from>
  -role <database administrator SQL role name>
  -trusted <use trusted authentication>
  -database <database to manage>
  -z

available commands:
  adding a new user:
    add <name> [ <parameter> ... ]
  deleting a current user:
    delete <name>
  displaying all users:
    display
  displaying one user:
    display <name>
  modifying a user's parameters:
    modify <name> <parameter> [ <parameter> ... ]
  changing admins mapping to RDB$ADMIN role in security database:
    mapping <set|drop>
  help:
    ? <interactive only>
    help
  displaying version number:
    z <interactive only>
  quit interactive session:
    quit <interactive only>

available parameters:
  -pw <password>
  -uid <uid>
  -gid <uid>
  -fname <firstname>
  -mname <middlename>
  -lname <lastname>
  -admin <yes|no>
```

Figura 3. Parâmetros do comando gsec



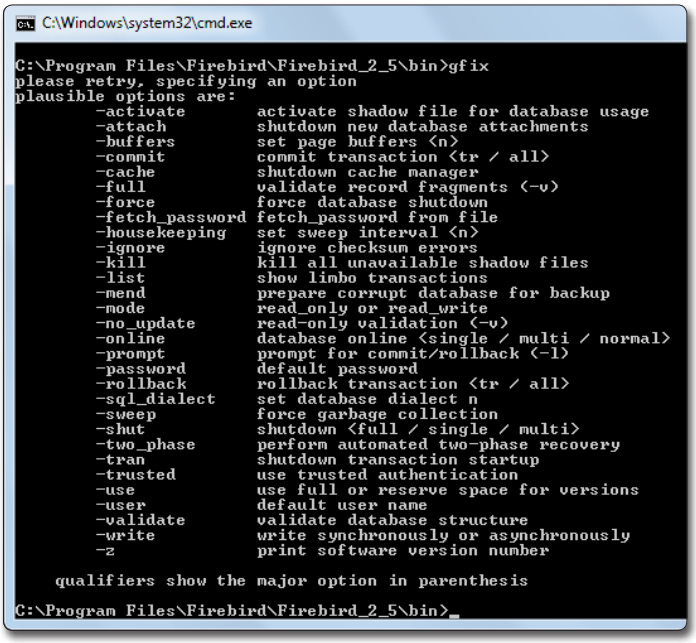


Figura 4. Parâmetros do gfix

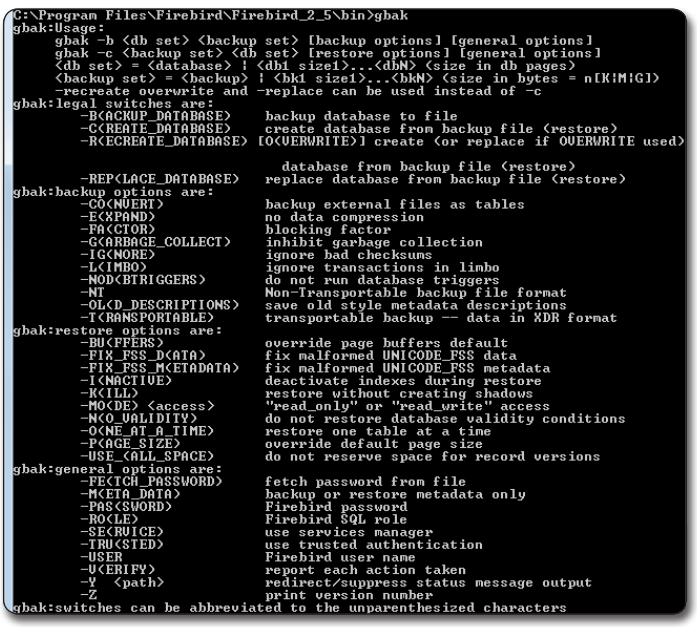


Figura 5. Parâmetros do gbak

Se for para substituir o arquivo de banco de dados antigo (não recomendo), o comando é:

```
gbak -r -rep -user SYSDBA -password masterkey c:\caminho_do_backup\nomebackup.fbk c:\caminho_do_banco\banco.fdb
```

nbackup

Este aplicativo serve para gerar um backup incremental do banco de dados. Para poder utilizar este aplicativo, é necessário uma política de backup muito bem planejado, por isso, não recomendo a utilização do mesmo. A **Figura 6** mostra os parâmetros permitidos pelo nbackup.

gstat

Este aplicativo traz diversas informações estatísticas de um banco de dados. As informações dependem dos parâmetros utilizados no momento que executar o aplicativo pela linha de comando do windos. Para saber os parâmetros, digite gstat -?. Veja na **Figura 7**, aonde mostra os parâmetros.

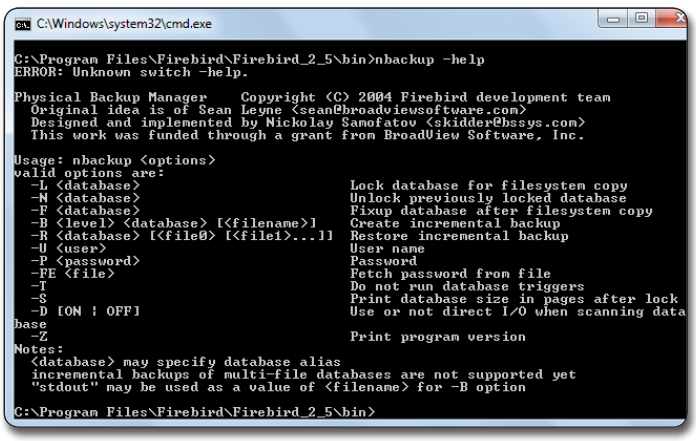


Figura 6. Parâmetros do nbackup

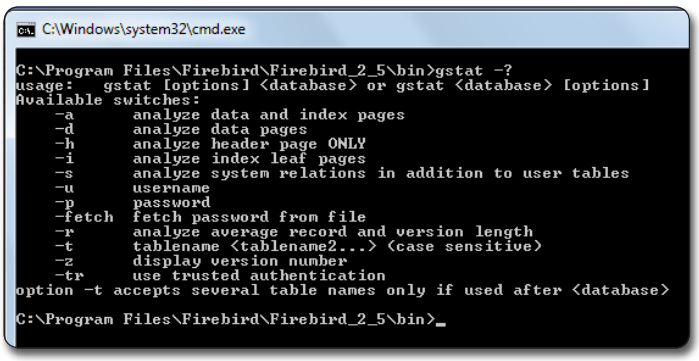


Figura 7. Lista de parâmetros do gstat

Tipos de servidores Firebird

O Firebird, até a versão 2.0, possuía três tipos de versões de servidores de banco de dados: Super Server, Classic e Embedded. A partir do Firebird 2.5 houve a criação de mais um tipo de banco de dados Firebird, a SuperClassic.

Veremos uma síntese de todas as versões a seguir:

• Classic: A arquitetura clássica utiliza-se da mesma concepção de outros bancos de dados, como Oracle, SQL Server, DB2, Sybase. Essa arquitetura é uma arquitetura que exige mais hardware! Para se ter uma ideia, essa arquitetura não utiliza por completo o conceito de multi-thread e thread-save, pois cada conexão com um banco de dados, será uma conexão independente, ou seja, será reservado um espaço de memória para cada conexão e cada conexão será um processo no servidor.

• SuperServer: A arquitetura SuperServer, é uma arquitetura que não exige muito hardware, pois todas as conexões com o banco de dados é compartilhado, ou seja, é compartilhado uma área de memória e caso alguém já tenha realizado alguma recuperação

de dados que tenha os dados que vou buscar, o servidor Firebird utiliza os mesmo dados. A principal diferença entre o SuperServer e o Classic está no fato do SuperServer basear-se em threads, ou seja, um único processo cria diversas threads, de acordo com o número de conexão com o banco de dados.

• Embedded: Esta arquitetura permite que possa distribuir as aplicações junto com o Firebird, sem a necessidade do Firebird ser instalado. Há a possibilidade de instalar o Firebird Embedded e a aplicação, num DVD para demonstração. Antes da versão 2.5 do Firebird, esta arquitetura tinha algumas limitações, como por exemplo, só podia conectar um usuário por vez no banco de dados. Isso foi possível, pois esta arquitetura baseou-se na arquitetura SuperClassic ou invés do SuperServer.

• SuperClassic: O SuperClassic é uma arquitetura híbrida das arquiteturas Classic e SuperServer. Ela é indicada para uso de máquinas multi-cores/SMP com grande quantidade de conexões.

Conclusão

O banco de dados Firebird é poderoso e possui os recursos que os demais bancos de dados convencionais possuem. Vale a pena investir e usar o Firebird, pois é um banco de dados que está em constante evolução e atende a mais de 90% das necessidades do mercado e isso a um custo zero, diminuindo o custo na implantação e no suporte do dia a dia, pois o Firebird não precisa de DBA para poder administrá-lo.

Todas essas características fazem o Firebird um banco de dados competitivo no mercado. Torna-se um evangelizador do Firebird, divulgando os recursos e o quanto é competitivo no mercado.

Use em quaisquer situações e momentos, use para armazenar dados de sistemas ERP, como também, como uma base, datacenter ou para utilização de sistemas BI.

Fabiano Moura
fabiano@fabianomoura.com.br
http://fabianomoura.com.br
Analista Desenvolvedor de Sistemas na Pongeluppe Informática há 5 anos. Desenvolve sistemas hospitalares e para a área da saúde, utilizando conceitos ERP. Graduado na FATEC Centro Paula Souza em Taquaritinga – SP e fazendo MBA em Gestão da Qualidade de Software na UNIARA, Araraquara – SP. Embaixador do MindTheBird, aonde possui a missão de evangelizar e divulgar o Firebird a todas as pessoas.

Fundação FirebirdSQL
http://www.firebirdsql.org

MindTheBird
http://www.mindthebird.com

Firebase
http://www.firebase.com.br

IBExpert
http://ibexpert.net/ibe/

IBPhoenix
http://www.ibphoenix.com/

IBSurgeon
http://www.ibsurgeon.com/

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Novidades do Delphi

Uma visão geral sobre as novid

Resumo DevMan

De que trata o artigo:

Este artigo é totalmente destinado às novidades do novo Delphi XE2. Assim como acontece a cada nova versão lançada, o IDE veio recheado de recursos novos, visando atender aos mais variados segmentos. Só para citar, são novidades nas áreas de aplicativos móveis, desenvolvimento multi-plataforma, relatórios, interfaces de usuário, estrutura, entre outras. Assim sendo, o artigo fará um overview sobre as principais, para que o leitor tenha a ideia exata sobre cada uma e assim se aprofundar nas que possam agregar valor ao seu cotidiano de desenvolvimento.

Em que situação o tema é útil:

As novidades do XE2, como já eram esperadas, não se restringem a um segmento. Dessa forma, desenvolvedores de áreas distintas poderão tirar proveito de diferentes formas. Por exemplo, para o desenvolvimento multi-plataforma, os destaques ficam por conta do suporte nativo a 64 bits e Mac. Ainda neste quesito, adiciona-se o tão falado FireMonkey, o novo framework da Embarcadero para a construção de aplicações visualmente mais ricas, que poderão ser executadas no Windows, MacOS X e iOS. Outra novidade muito esperada é o LiveBindings, o novo recurso para ligação de dados que tem seu início nessa nova versão do Delphi.

Novidades do Delphi XE2:

FireMonkey, LiveBindings, VCL Styles e FastReport são alguns dos nomes que serão vistos dentre as novidades dessa nova versão do Delphi, a XE2. Todas elas dizem respeito a determinado segmento do desenvolvimento como, por exemplo, o VCL Styles, que vem para suprir uma necessidade antiga dos projetos de aplicações Delphi Win32 e que até então só eram atendidas através de componentes de terceiros, para estilização visual da própria aplicação. Agora este tipo de recurso se tornou nativo. Este e outros dos principais recursos da nova versão do IDE da Embarcadero serão apresentados no decorrer do artigo. Além das apresentações conceituais necessárias, na parte prática do artigo serão efetuados dois exemplos que mostram um pouco do processo de desenvolvimento com alguns destes novos recursos.

A nova versão XE2 lançada recentemente torna o Delphi ainda mais poderoso em termos tecnológicos. Isto porque, sua tão conhecida e eficiente produção de aplicativos para a plataforma Windows 32 bits, agora ganha novos rumos. O suporte ao Windows 64 bits e Mac são a bola da vez e que abrem novos horizontes para a comunidade de desenvolvedores, ao mesmo tempo em que a produtividade e excelência na forma de desenvolvimento são devidamente mantidas. Prova disso é o poder RAD que consagrou a ferramenta, que mais uma vez se estabelece, refletindo em novos grupos de componentes para os mais diversos nichos da programação. Sem perder tempo, a seguir é feita uma apresentação das principais novidades, além de alguns exemplos práticos como complemento.

Unit Scope Names

Antes mesmo de apresentar os novos recursos dessa versão, é importante se fazer uma ressalva à nova forma de nomeação das units utilizadas no IDE. Esta novidade

XE2
ades dessa versão

Fabrício Hissao Kawata

é conhecida como Unit Scope Names, isto porque agora as units são prefixadas com os nomes das units das bibliotecas as quais se referem: VCL, RTL, além da nova FMX (que será vista adiante). Exemplificando, num projeto VCL Forms no XE2, na seção uses de uma eventual unit, ao invés SysUtils e Dialogs, encontraremos algo como System.SysUtils e VCL.Dialogs. Tomando como base este último, VCL é o nome do escopo da unidade, enquanto que Dialogs é o nome da unit. Com esta nova convenção, a intenção da Embarcadero é, entre outras coisas, poder classificar as units em grupos e assim poder diferenciar os membros de units distintas, mas que possuem nomes iguais.

Nota do DevMan

RAD (Rapid Application Development) é um conceito que se tornou muito popular entre as ferramentas de desenvolvimento, em especial o Delphi. Sua premissa consiste em diminuir o tempo de construção de um software oferecendo recursos avançados de automação. Entre esses recursos está o uso de componentes visuais prontos que já encapsulam um comportamento próprio, estando prontos para o uso, e o forte favorecimento da programação orientada a eventos em sua arquitetura. Para um desenvolvedor iniciante, é muito fácil e rápido arrastar e soltar componentes visuais já prontos e simplesmente responder programaticamente aos seus possíveis eventos como um clique de mouse ou a perda de um foco de digitação.

Compilador 64 bits

Agora é possível desenvolver aplicações Windows 64 bits com o Delphi. Este recurso sem dúvida nenhuma era um dos mais esperados pela comunidade, visto sua notória menção em fóruns de discussão e outros pontos da Web. Além disso o DCC64, que é o nome do seu novo compilador de linha de comando, apresenta algumas novas características peculiares voltadas a 64 bits, tais como índices de arrays dinâmicos e ponteiros. Em termos de estrutura, no total são três as bibliotecas que suportam Win32 e Win64: as tradicionais VCL e RTL e a nova FMX.

Para recapitular, a VCL ou Visual Component Library é a biblioteca de objetos de componentes visuais do Delphi, utilizada para a construção das interfaces de usuário em aplicações Windows. Para tal, ela oferece um grande número de componentes visuais e não visuais, tais como os da paleta Standard, dbExpress, entre outros. Já a RTL ou Run-Time Library é a biblioteca processual (de rotinas e classes base) do Delphi. Seu conjunto de units inclui as tradicionais System, SysUtils, Classes e IOUtils. Nestas estão inseridas classes como TObject, TPersistent e TComponent, além de rotinas globais e outras classes utilitárias que provêem as funcionalidades necessárias e comuns a outras bibliotecas, como a VCL e também à nova FMX. Em termos de plataforma, a RTL é suportada em aplicações Windows (32 e 64 bits) e Mac (MacOS X).

FMX ou FireMonkey é a mais nova biblioteca de componentes visuais inserida no Delphi XE2, utilizada para criar aplicações para as plataformas Windows (Win32 e Win64) e MacOS X. Ela contém elementos para a construção de interfaces gráficas sofisticadas e estilizadas, além de animações, conforme será visto adiante.

No XE2 as tradicionais VCL e RTL foram redefinidas de forma a trabalhar com as aplicações 64 bits da mesma maneira como já se trabalhava com as aplicações 32 bits. Com isso, a Embarcadero visa o reaproveitamento de código-fonte produzido, uma vez que, se você tiver uma aplicação que faz uso especificamente de VCL e RTL, poderá utilizar seu código tanto na plataforma Win32 quanto na Win64. Ainda sobre estas aplicações, caso elas façam uso de componentes visuais, será necessário a compilação de dois aplicativos separados direcionando seu destino para a plataforma condizente. Este direcionamento é provido pelo próprio IDE, dentro do Project Manager, através de um novo recurso denominado Target Plataforms, conforme pode ser visto na **Figura 1**. Dessa forma, através de um simples duplo-clique, é possível trocar a plataforma de destino a qualquer momento durante o desenvolvimento nessa nova versão do Delphi.

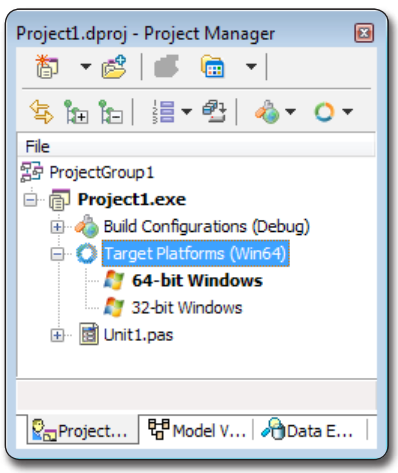


Figura 1. Target Plataforms

Um detalhe importante a ser mencionado sobre o desenvolvimento 64 bits no IDE é com relação a três palavras comuns na construção de um aplicativo: execução (run), depuração (debug) e implantação (deploy). Ao se optar por esta plataforma, é neces-

sário se atentar ao tipo de ambiente de desenvolvimento. Entenda ambiente como sendo a máquina (PC) e o sistema operacional (Windows) utilizados. Este sistema em questão deverá essencialmente ser 64 bits ou estar ligado a um. Portanto, se a máquina em questão for 64 bits, rodando em um sistema também 64 bits, sua execução, depuração e implantação se darão da mesma forma que era realizada até então, para as aplicações tradicionais Win32. Todavia, caso a máquina esteja usando um sistema 32 bits, será necessário instalar e executar um servidor de aplicativo remoto no sistema destino (64 bits), denominado Plataform Assistant (seu instalador já acompanha o Delphi XE2), estabelecendo assim a conexão do sistema de desenvolvimento com um sistema 64 bits assistente. Especificamente em termos de debug, o Plataform Assistant já instala todos os arquivos necessários para a depuração de aplicações multi-plataforma. Para o deploy dessas aplicações em sistemas remotos é disponibilizado ainda um gerenciador, denominado Deployment Manager, que trabalha em conjunto com o Plataform Assistant. Através dele é possível visualizar, adicionar, editar e excluir os arquivos que fazem parte do deploy da aplicação, da mesma forma que em aplicações Win32.

FireMonkey

FireMonkey (ou simplesmente FMX) é o nome do novo framework de desenvolvimento, que acompanha o novo Delphi XE2, voltado para a construção de aplicativos visualmente atraentes e multi-plataformas. Pelo fato de aproveitar os recursos da unidade de processamento gráfico da máquina, a GPU, o framework produz como resultado aplicações com um visual mais moderno, se comparado às tradicionais aplicações VCL Win32. Além disso, possibilita ainda a criação de animações e oferece suporte a estilização. Por estes motivos, muitos o comparam à tecnologia WPF (Windows Presentation Foundation) da Microsoft, que inclusive já foi tema de outros artigos nesta revista. Com o FireMonkey, a Embarcadero visou atingir não só sua plataforma padrão Windows (32 e 64bits), como também atender o seu novo suporte à plataforma Mac, produzindo aplicativos para MacOS X (Desktop), e iOS (Mobile). Na **Figura 2** podem ser vistas os novos tipos de aplicação, voltadas ao desenvolvimento FireMonkey.

Dentre as principais características dessas aplicações, os destaques ficam por conta de:

- Suporte a gráficos 2D e formas primitivas (com recursos de brushes, pen, transforms, etc);
- Suporte a gráficos 3D;
- Suporte a animações. Estas são otimizadas a fazer o mínimo uso da CPU, além da correção automática de frame rate (taxa de quadro);
- Robusta vector engine, para trabalho com vetores, assim como acontece no Microsoft Windows Presentation Foundation (WPF) e Macromedia Adobe Flash;
- Robusta GUI engine, para trabalho com elementos de interface de usuário (Ex: Button, TextBox, Memo, ListBox, etc);
- Avançada Skinning engine, baseada em temas e estilos de gráficos vetoriais;
- Gráficos vetoriais independente de resolução;
- Proporciona uma enorme flexibilidade em relação aos elementos de interface (layout), permitindo a composição de formas e controles;
- Multi-plataforma, com suporte às plataformas Windows (32 bits e 64 bits) e Mac (MacOS X e iOS). Trabalha com uma camada que abstrai os recursos de cada sistema operacional como, por exemplo, suas janelas e caixas de diálogo;
- Suporte aos formatos de imagem JPEG, GIF, PNG e TIFF;
- Suporte a Unicode, que permite a manipulação consistente de texto em qualquer sistema de escrita;
- Designer e editor de propriedades ao estilo WYSIWYG (What You See Is What You Get), termo este, que traduzido para o português seria algo como "O que você vê é o que você obtém". É a capacidade de se definir elementos da interface visual ao mesmo tempo em se permite a visualização de tais mudanças;

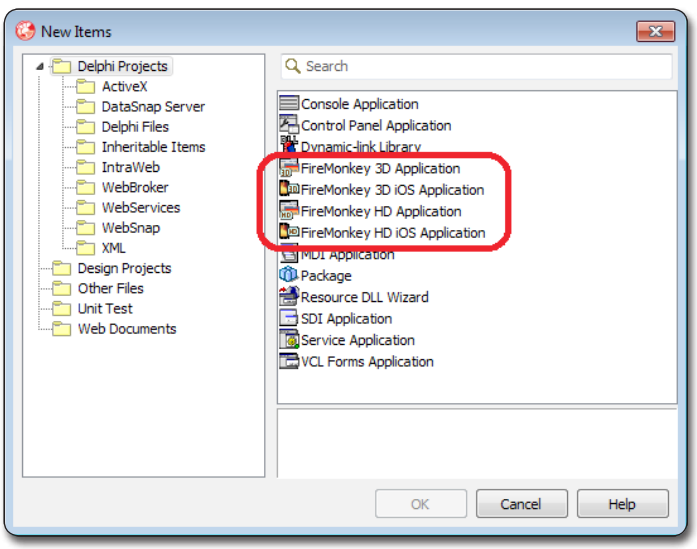


Figura 2. Novos tipos de aplicação para FireMonkey disponíveis no IDE

Uma observação importante a se fazer e que a própria Embarcadero faz questão de ressaltar, é que o FireMonkey e a VCL são incompatíveis, em vista da diferença que há entre os frameworks. Em outras palavras, não se pode usar ambas num mesmo projeto. Neste caso, você deverá optar pela escolha entre um tipo de projeto específico. Até mesmo por questões de diferenciação, as units do FireMonkey são prefixadas com FMX (Ex: FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs, etc).

No desenvolvimento com o FireMonkey, entre os tantos diferenciais apresentados em relação ao desenvolvimento VCL, está o seu trabalho com acesso a banco de dados. Esta conectividade se dará pelos tradicionais frameworks dbExpress e DataSnap, no entanto, o FMX utiliza-se do novo recurso de LiveBindings, para se estabelecer a ligação entre seus controles e os componentes de dados.

Desenvolvimento para Mac (OS X)

Ainda sobre o assunto multi-plataforma, o novo XE2 também possibilita a criação de aplicações destinadas à plataforma Macintosh (ou simplesmente Mac). Mais especificamente essas aplicações irão rodar sobre sistema operacional MacOS X. Assim como acontece para a plataforma 64 bits, o Delphi apresenta um novo tipo de compilador específico para a plataforma Mac, denominado DCCOSX.

Para que este tipo de desenvolvimento seja possível será necessário, além da própria máquina de desenvolvimento, um hardware complementar específico. Na prática, o ambiente de desenvolvimento deverá ter uma máquina Windows que possua o Delphi XE2 instalado, devidamente conectado a uma máquina Mac através de uma rede. Assim como acontece com a plataforma 64 bits, também é necessária a instalação do Plataform Assistant na máquina de destino, ou seja, no Mac.

Na plataforma Win32, o tipo de aplicação mais simples e comum no desenvolvimento Delphi é a do tipo console e esta é também devidamente suportadas pela plataforma Mac, ao contrário das aplicações VCL que continuam sendo exclusivas para Windows. Este tipo de detalhe fica mais evidente em relação às opções de Target Plataforms disponíveis em cada tipo de projeto. Na **Figura 3**, que reflete as opções de uma Console Application, em sua Target Plataforms fica disponível para adição o novo tipo de plataforma OS X. Assim como acontece para as aplicações Console, o novo tipo de aplicação denominado FireMonkey HD Application também oferece suporte à plataforma Mac e sua configuração é feita da mesma forma.

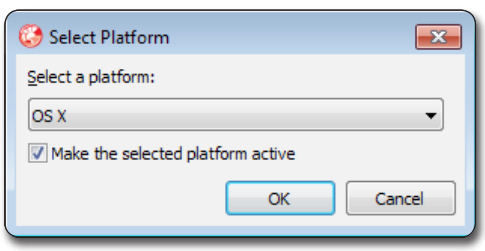


Figura 3. Selecionando a plataforma OS X, como plataforma destino

Na parte estrutural, a RTL do Delphi oferece funcionalidades básicas já disponíveis para a plataforma Windows também para Mac. Um exemplo clássico é a unit SysUtils.pas (que agora é prefixada, ficando nomeada como System.SysUtils.pas). Soma-se a isso, outro grupo de units exclusivas para este tipo de desenvolvimento e cujos nomes são prefixados com Macapi, em referência a Mac API (Ex: Macapi.OpenGL, Macapi.Consts, Macapi.Foundation, etc). Algumas dessas, inclusive, são utilizadas pelo framework FireMonkey, já que também pode ser utilizado na plataforma Mac. Estas units provêem interfaces Delphi para frameworks Mac escritos em Objective-C. Já a VCL, como dito, não está disponível para Mac.

Com relação a tratamento de exceções, a unit SysUtils apresenta algumas classes de exceção, derivadas de EExternal, que representam as exceções não provindas da linguagem. No Mac uma exceção EExternal é originada como sendo uma exceção Mach, que por sua vez faz com que a RTL da SysUtils crie uma thread exclusiva para as notificações de exceção do sistema operacional. Dessa forma, ao receber a notificação de uma exceção Mach, esta é transformada e levantada como uma exceção correspondente na linguagem Pascal.

LiveBindings

LiveBindings é o nome do novo recurso apresentado pelo Delphi XE2, para ligação de dados, baseado em expressões. Graças a elas é possível ligar objetos distintos por meio de suas propriedades. Esta característica é a mais recente aposta da Embarcadero, tanto que ela é suportada tanto pela VCL tradicional quanto pelo novo FireMonkey. As expressões utilizadas pelo LiveBindings são chamadas de expressões relacionais ou de ligação, que são utilizadas ao se estabelecer a ligação entre objetos de origem (source object) e destino (também tratado como objeto de controle – control object), numa relação unidirecional ou bidirecional.

Nota do DevMan

Objective-C, ou simplesmente Obj-C é uma linguagem de programação orientada a objetos utilizada no MacOS X. Ela foi construída sobre a linguagem C, tendo sua sintaxe derivada também do Smalltalk. Dado seu histórico, ela foi criada por Brad Crox e Tom Love, ainda na década de 80.

Nota do DevMan

Existem várias variações da linguagem de desenvolvimento Smalltalk. A primeira se tornou pública em 1980 e até existem versões que são atualizadas. Smalltalk é uma linguagem puramente orientada a objetos, simples e uniforme. Ela influenciou a maioria das linguagens orientadas a objeto modernas. Na sua utilização de um modelo de objetos, as seguintes características são empregadas:

- Tudo é um objeto;
- Objetos se comunicam por passagem de mensagens;
- Classes descrevem o estado e comportamento dos objetos que elas geram;
- Métodos são públicos;
- Variáveis de instância são privadas;
- Classes herdam de apenas uma classe, não há herança múltipla.

Na prática, um objeto pode ser vinculado a outro ou a si mesmo através de uma ou mais propriedades. Para que isso aconteça, é necessária simplesmente a definição de uma expressão de ligação. Dessa forma, num exemplo clássico, podemos estabelecer a ligação entre um TEdit e um TLabel, de forma que o texto digitado na caixa de texto seja automaticamente mostrado no label. Ou ainda um TSpinEdit ligado a um TShape, onde o valor informado no primeiro irá refletir diretamente no valor da altura ou largura do segundo. Neste tipo de vínculo estabelecido, podemos perceber uma das características do LiveBindings, que é a capacidade de alterar as propriedades dos objetos envolvidos. As alterações realizadas no objeto origem (TEdit) acabam por alterar a propriedade Caption do objeto destino (TLabel).

No IDE o LiveBindings ganhou uma guia exclusiva na paleta de componentes (Tool Palette), conforme mostra a **Figura 4** para VCL e a **Figura 5** para o FireMonkey. Nela, podemos ver os três novos componentes visuais disponíveis, que auxiliam na definição de LiveBindings entre os componentes da VCL e FireMonkey, em tempo de projeto. Isso quer dizer que você poderá arrastar e soltar cada um deles para um formulário e assim manuseá-los, no melhor estilo Delphi. A seguir, por questão de conhecimento, segue um pequeno resumo de cada componente:

- TBindingsList: funciona como uma espécie de gerenciador de bindings, que controla a lista de bindings definidos na aplicação. Lembra muito um TActionList, mas ao invés de ações, permite criar, editar, excluir e visualizar bindings;
- TBindScope: é o componente utilizado no binding de componentes;
- TBindScopeDB: utilizado no binding com banco de dados;
- TBindNavigator: disponível para FireMonkey. Este componente, como o próprio nome sugere, é um componente similar ao já conhecido TDBNavigator, utilizado na manipulação e navegação de dados.

Na parte prática do artigo, você poderá conferir como criar um simples LiveBinding entre dois objetos, num tradicional "exemplo clássico".

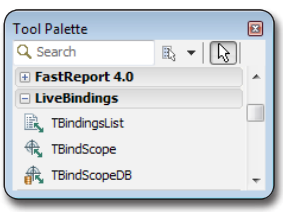


Figura 4. LiveBindings na Tool Palette - VCL

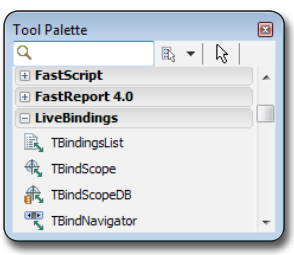


Figura 5. LiveBindings na Tool Palette – FireMonkey

DataSnap Mobile Connectors

Agora no XE2 é possível criar aplicações para dispositivos móveis (tais como iOS e Windows Phone) fazendo uso dos novos conectores DataSnap Mobile, os chamados DataSnap Mobile Connectors. Eles são disponibilizados em um framework que é utilizado para acessar serviços de um servidor DataSnap REST.

Através dos diversos DataSnap Mobile Connectors disponibilizados pelo novo Delphi, tem-se o suporte às seguintes plataformas móveis:

- iOS 4.2 (linguagem Objective-C);
- Android 2.1 (linguagem Java);
- BlackBerry (linguagem Java, SDK 5 e SDK 6);
- Windows Phone 7 (linguagem C#, tecnologia Silverlight).

A invocação de métodos do servidor se dá por meio das tradicionais classes Proxy produzidas pelo gerador de proxy DataSnap (Proxy Generator). Isto é possível devido a uma nova Proxy Dispatcher (expedidor de proxy) introduzida, que trabalha em conjunto com o recurso de Metadata Provider e os componentes de Server Class existentes. Esse Dispatcher é quem proverá ao desenvolvedor a obtenção, em uma máquina remota, do Proxy gerado no servidor. Isto acontece, pois este tipo de desenvolvimento para aplicativos móveis não é realizado diretamente dentro do Delphi.

Dito isto, na prática, o primeiro passo para o desenvolvimento móvel com DataSnap é preparar o Proxy Dispatcher no servidor, de forma que ele possa gerar o proxy adequado para a plataforma de destino. Um exemplo básico seria a construção de aplicativos para Android, que faz uso da linguagem Java. Nesta situação, o proxy desejado é representado pelo arquivo DSProxy.java contendo o código gerado pelo servidor DataSnap. Ainda como complemento, o Proxy gerado pode necessitar de alguns arquivos adicionais que, basicamente, contém os tipos de dados suportados pelo DataSnap no Delphi. Só para citar, especificamente para a linguagem Java, estes arquivos são agrupados em um pacote exclusivo denominado com.embarcadero.javaandroid.

Para facilitar ainda mais o desenvolvimento, nos próprios Wizards DataSnap foi inserida a opção para que se indique o trabalho com os Mobile Connectors, conforme visto em destaque na **Figura 6**.

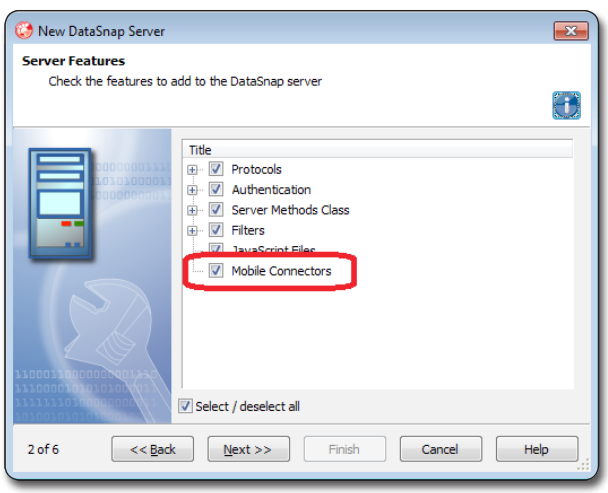


Figura 6. Opção que habilita o trabalho com Mobile Connectors

Pelo simples marcar deste checkbox o Wizard aplica algumas configurações específicas no projeto do servidor DataSnap que se está criando. A primeira é criar um novo diretório no projeto, denominado Proxy. Além disso, é gerado um evento pós-build, que faz a cópia dos arquivos estáticos do projeto (como exemplo os arquivos JavaScript, CSS e proxy) para o local de saída do mesmo, prevendo assim o seu deploy. Depois, ainda são adicionados três componentes já pré-configurados:

• Proxy Generator (TDSProxyGenerator), que será responsável exclusivamente pela geração do proxy;
• Metadata Provider (TDSRestMetaDataProvider ou TDSServerMetaDataProvider). Este componente irá expor os metadados do servidor ao Proxy Generator;
• Proxy Dispatcher (TDSProxyDispatcher ou TDSHTTPServiceProxyDispatcher) processa os requests, ou seja, os pedidos relacionados a um proxy específico.

Como são várias as opções de plataformas móveis suportadas, o recomendado pela Embarcadero é que se adicione à cláusula uses de cada gerador de proxy, a unit relativa à plataforma destino. Neste caso as opções disponíveis são: DSProxyObjectiveCiOS, DSProxyJavaAndroid, DSProxyJavaBlackBerry e DSProxyCsharpSilverlight.

Neste processo onde o servidor DataSnap irá gerar um proxy para uma linguagem em particular, o arquivo é gerado dentro de uma pasta específica (csharp_silverlight, objective_ios42, java_blackberry, e java_android) no diretório "proxy" criado, onde se junta aos demais arquivos necessários. A partir de então o próprio servidor faz a compressão da pasta, gerando um arquivo .zip, que é o que deverá ser baixado pelo cliente.

No cliente, para se baixar este arquivo zipado, a maneira mais simples é acessando diretamente a URL do servidor no formato padrão "endereco_servidor:porta/proxy/arquivo.zip". Com ele em mãos, tem-se então todo o código-fonte necessário para que a plataforma móvel possa invocar devidamente os métodos expostos pelo servidor DataSnap. Alternativamente o Delphi disponibiliza ainda outras duas opções para se efetuar o download do arquivo, através de aplicações de linha de comando Win32 e Java, denominadas Win32ProxyDownloader.exe e JavaProxyDownloader.java respectivamente. Elas acompanham a instalação do IDE e ficam localizadas em sua pasta bin. Na prática, a chamada a estes executáveis utilitários permite a inclusão de parâmetros complementares que representam as diversas opções disponíveis. Por exemplo, para o executável Win32ProxyDownloader, a sintaxe padrão de comando é basicamente a seguinte: Win32ProxyDownloader.exe -protocol -host -language -output. Seguindo a ordem dos parâmetros, o primeiro refere-se ao protocolo utilizado (HTTP ou HTTPS). Host representa o nome do host e a porta do servidor. A única opção obrigatória dentre todas, Language faz menção ao nome do identificador da linguagem utilizada na plataforma destino. Por fim, Output informa o caminho do local onde o proxy será salvo. A seguir é mostrado um exemplo real de sua utilização:

Win32ProxyDownloader.exe -host localhost:8060 -language java_android -output c:\proxy

VCL Styles

VCL Styles permite a alteração da aparência de uma aplicação VCL através do uso de estilos. Fazendo um comparativo, em uma simples página Web HTML, os estilos são utilizados para a definição da apresentação visual do documento. De forma semelhante, no Delphi XE2, um estilo pode ser definido como um conjunto de detalhes e especificações visuais que irão definir a aparência da aplicação, semelhante à mecânica de temas do Windows.

Para que se entenda a forma de trabalho, internamente falando, do VCL Style, é importante ter em mente que cada controle VCL é composto por dois itens principais: parte e estado. Por exemplo, um simples componente TScrollBar é formado por diversas partes, ou seja, suas peças. Neste caso, a olho nu podemos notar a presença de um controle deslizante (slider) principal e dois botões laterais indicadores de direção. Da mesma forma, em um TSpinEdit vemos uma caixa de entrada de valores e outros dois botões indicadores de direção (para cima e para baixo). Em ambos os componentes, cada um de seus botões deverá possuir alguns estados, tais como normal e pressionado. Dito isto, o que um VCL Style faz para alterar a aparência de um controle é definir valores específicos a cada parte e estado deste.

Anteriormente ao VCL Styles, uma prática comum em projetos Win32 Delphi que necessitavam de algum suporte a estilização, era o trabalho com componentes de Skins de terceiros. Basicamente cada novo estilo ficava salvo em um arquivo específico que deveria ser carregado pela aplicação para que a alteração visual fosse estabelecida. Dessa forma, o próprio usuário poderia alterar o aspecto visual da aplicação ao seu gosto. Agora este tipo de recurso é nativo, o que facilita muito o trabalho do desenvolvedor. Para que você tome conhecimento do resultado final da utilização de VCL Styles, a Embarcadero disponibiliza junto à instalação do Delphi alguns arquivos prontos de estilo, que já podem ser devidamente aplicados em qualquer projeto VCL.

A maneira mais fácil de visualizar a utilização destes estilos numa aplicação corrente é através do próprio IDE, na também nova opção de projeto Application Appearance. Conforme pode ser visto na **Figura 7** são disponibilizadas por padrão cinco opções de estilos, além da tradicional Windows. Já neste momento, para cada item da lista, é possível se visualizar uma prévia de suas características, através do botão Preview. Ao marcar alguma das opções, para que a aplicação corrente instantaneamente faça proveito do estilo desejado, basta selecioná-lo no combo Default style. Lembrando que essa estilização é perceptível somente em tempo de execução.

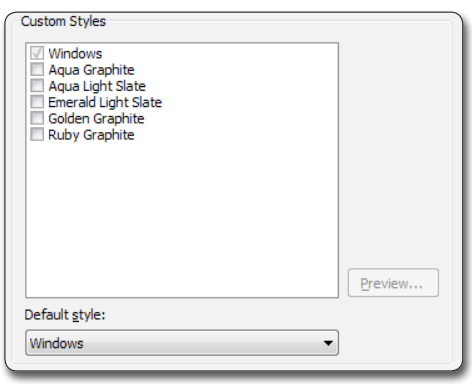


Figura 7. Estilos disponíveis, por padrão, no IDE

A **Figura 8** mostra o estilo padrão e a **Figura 9** exibe um estilo aplicado. Como se percebe, na primeira imagem é mostrada a aplicação em sua forma tradicional, tirando proveito dos recursos disponibilizados

até então, em termos visuais, para aplicações VCL. Na segunda imagem a mesma aplicação faz proveito do novo recurso de VCL Style, fazendo com que sua aparência visual mude por completo.

Não deixando de mencionar, é possível também criar seus próprios estilos ou modificar algum existente, se for o caso. Para tal, foi incorporado ao Delphi uma nova ferramenta denominada VCL Style Designer, encontrada no menu Tools do IDE.

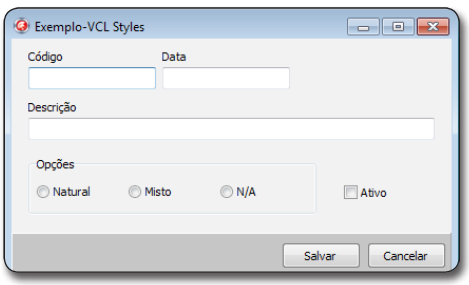


Figura 8. Aplicação VCL padrão

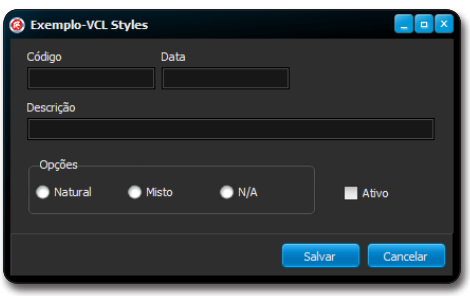


Figura 9. Aplicação VCL fazendo uso de VCL Style

FastReport

Até sua versão anterior, a XE, o Delphi trazia consigo uma única opção para relatórios, o já conhecido Rave Reports. Com o lançamento do XE2 uma nova forma de geração rápida e eficiente de relatórios foi disponibilizada, graças à nova ferramenta FastReport, que agora se integra ao IDE, disponibilizando uma nova guia de componentes na Tool Palette (**Figura 10**).

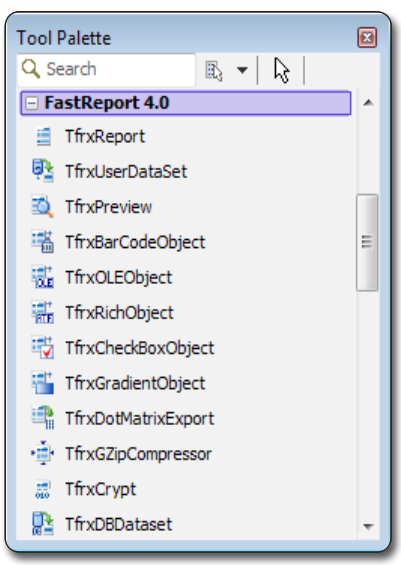


Figura 10. Componentes FastReport

Esta integração se deve muito ao fato do FastReport possuir um histórico muito positivo perante o Delphi, uma vez que já era muito bem quisto e utilizado pela comunidade.

Abrindo projetos antigos no XE2

O Delphi sempre teve uma boa fama na questão da retro-compatibilidade que é mantida a cada nova versão que surge do IDE. Tomando como exemplo a versão anterior, XE, nela poderíamos facilmente "importar" um projeto legado, construído em Delphi 7. Neste caso qualquer alteração que se faça necessária no projeto, o próprio IDE se encarrega de notificar e auxiliar o desenvolvedor para o processo. Da mesma forma o XE2 mantém esse mesmo tipo de funcionalidade, dando a opção para se converter e renomear o novo projeto, ou ainda, converter e realizar o backup do projeto original. As configurações relativas a esse tipo de "migração" podem ser manipuladas dentro do Project Upgrading, presente em Environment Options.

Pelo fato do Delphi XE2 possibilitar a escolha da plataforma destino de um projeto, algumas exceções são estabelecidas, como a impossibilidade de se direcionar MacOS X como sendo a plataforma de destino para um projeto que está sendo importado. Diferentemente, é possível que um projeto Win32 seja importado e direcionado para a plataforma 64 bits, no entanto, com ressalvas. Isto é um alerta que a própria Embarcadero faz questão de mencionar, apontando alguns problemas pontuais como, por exemplo, operações que envolva ponteiros. Isto porque, para a plataforma 32 bits, o tamanho de um ponteiro é de 4 bytes, enquanto que na plataforma 64 bits esse valor é duplicado (8 bytes).

A seguir, para fechar o artigo com "chave de ouro", e até como uma forma de degustação de todas essas novidades apresentadas nessa nova versão XE2 do Delphi, serão construídos dois exemplos práticos.

Aplicação de exemplo 1 – LiveBindings

Para um bom entendimento de um assunto novo nada melhor que vê-lo e fazê-lo na prática. Acredito que uma das maiores curiosidades da comunidade, dentre as novidades dessa versão XE2 do Delphi, seja sobre LiveBindings. Portanto, conforme mencionado anteriormente, veremos como se dá a criação de um simples LiveBinding entre dois objetos.

Os objetos em questão serão dois componentes, TEdit e TLabel, presentes em uma aplicação VCL Forms Application tradicional. O objetivo é fazer com que o texto informado no TEdit seja instantaneamente mostrado no TLabel. Neste momento já identificamos o nosso objeto de origem (TEdit) e o de controle (TLabel). Neste último é onde deverá ser criado um novo binding, do tipo TBindExpression, através de sua propriedade LiveBindings. Ao fazer isso, automaticamente o IDE já adiciona um componente TBindingsList no formulário. Uma vez criado, é necessário informar ao novo LiveBinding o objeto de origem (Source Component) que trata-se do TEdit. Como já citado anteriormente, esse processo de binding trabalha com uso de expressões, portanto, é necessário informá-las ao novo LiveBinding criado. Isso se dará pelas suas

propriedades ControlExpression e SourceExpression. A primeira faz referência à expressão do objeto de controle que, neste caso, será a propriedade que queremos que seja atualizada, ou seja, Caption. Da mesma forma, a segunda refere-se à expressão do objeto de origem, portanto, informamos Text, que é o nome da propriedade de TEdit que queremos que sirva de fonte de dados ao objeto de controle.

Finalmente, é necessário que qualquer digitação de texto ocorrida na caixa de entrada, ou seja, no objeto de origem, seja notificada ao objeto de controle TLabel. Para isso, no evento OnChange de TEdit, digite a linha de código a seguir:

```
procedure TForm1.edtTextoChange(Sender: TObject);
begin
  BindingsList1.Notify(edtTexto, 'Text');
end;
```

O detalhe neste código é a chamada ao método Notify de TBindingsList para que o mecanismo de expressão envolvido seja notificado e assim o Binding seja avaliado. Uma observação pessoal que faço é que num primeiro momento esta "notificação" causa certa estranheza, uma vez que a ligação entre os objetos já foi efetivamente definida anteriormente. No entanto, isto é justificado pelo fato do Delphi não possuir um sistema de notificações autônomo, que se encarregue de informar à engine de expressão sobre as alterações ocorridas nas propriedades dos objetos envolvidos. Dessa forma, isto é feito de maneira manual em seus manipuladores de eventos. O esboço do resultado final desta aplicação de exemplo é visto na **Figura 11**, onde conforme o texto é digitado na caixa de entrada, o mesmo é replicado dinamicamente no label.

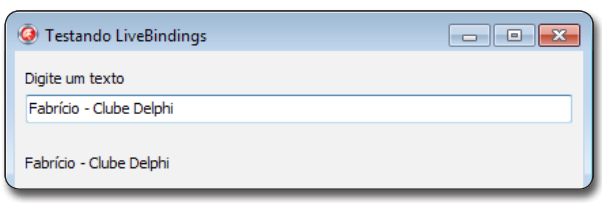


Figura 11. Aplicação de exemplo – Live Bindings

Aplicação de exemplo 2 – FireMonkey HD Application

Este próximo exemplo, apesar de simples, faz o mix de duas das principais novidades: FireMonkey e LiveBindings. Nesta demonstração a aplicação conterá um único formulário responsável em manter os dados de uma tabela de uma base de dados. Entenda como "manter" as atividades de visualização, inclusão, edição e exclusão dos registros. Visando facilitar ao máximo a base de dados será representada por um arquivo local, mais precisamente um arquivo .cds que acompanha a própria instalação do Delphi. Por padrão, estes arquivos de dados ficam localizados em C:\[NomeUsuario]\Public\Documents\RAD Studio\9.0\Samples\Data.

Portanto, a etapa inicial é a criação do novo tipo de projeto FireMonkey HD Application. Para a parte de dados, como a aplicação trabalhará com um arquivo local .cds, exclusivo para ClientDataSet, apenas um controle deste tipo será o suficiente para a manipulação das informações. Através de sua propriedade FileName indicamos o nome do arquivo de dados presente no diretório mencionado anteriormente. Dentre as várias opções disponíveis, para este exemplo foi selecionado o arquivo employee.cds.

Visualmente falando, o formulário deverá ter a aparência conforme mostra a **Figura 12**. Através dela pode-se notar a presença de elementos tradicionais como TEdit e TLabel, além do novo TBindNavigator que, como o próprio nome sugere terá a mesma funcionalidade de um TDBNavigator, para a manipulação e navegação pelos dados da aplicação.

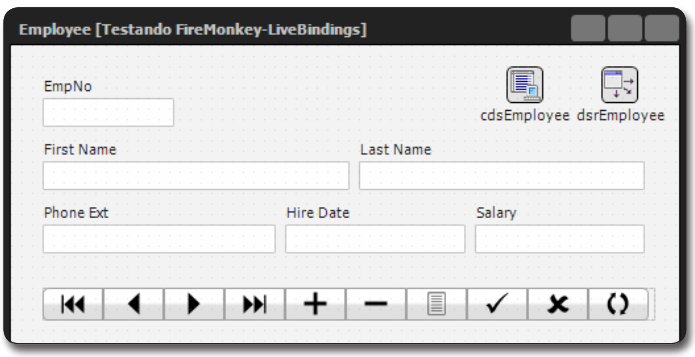


Figura 12. Visual da aplicação de exemplo

A etapa complementar diz respeito à ligação dos campos da tabela com os controles em tela. No FireMonkey, conforme já mencionado, isto se dá através dos novos recursos de LiveBindings. Novamente, para exemplificar como isto é feito, tomemos como base o TEdit para o campo EmpNo. Em suas propriedades há uma denominada LiveBindings que, ao ser clicada, oferece duas opções de escolha, New Live Bindings e Link to DB Field, conforme mostra a **Figura 13**. Conforme a necessidade do exemplo, a segunda é a indicada.

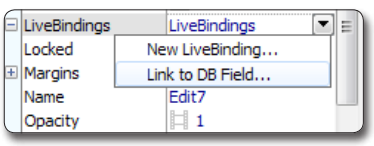


Figura 13. LiveBindings – Link to DB Field

Ao selecioná-la, abre-se então uma janela com a lista de campos disponíveis (**Figura 14**), bastando apenas escolher aquele condizente com a situação, neste caso, o campo EmpNo estabelecendo assim um novo DB Link. Dessa forma, este é o processo que deverá ser feito aos demais controles em tela que necessitam estarem vinculados a um campo da tabela de dados.

Ao se definir este tipo de Binding, o IDE automaticamente adiciona dois novos componentes no formulário: um TBindingList e um TBindScopeDB. De forma sucinta o primeiro representa a coleção de Bindings definidas na aplicação enquanto que o segundo é o componente que define um DataSource como sendo a fonte de dados para o Binding definido.

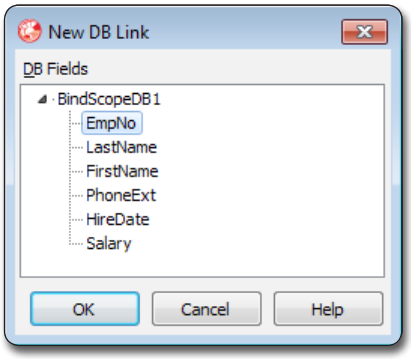


Figura 14. Novo DB Link

Para finalizar, basta apenas vincular o componente BindNavigator ao TBindScopeDB, para que ele tome conhecimento da fonte de dados que está sendo utilizada. Ao rodar a aplicação, logo nos testes iniciais, é possível perceber que todos os recursos do Navigator estão funcionando devidamente, ou seja, podemos inserir, excluir, editar e navegar pelos registros da tabela. Conforme a premissa do FireMonkey de produzir aplicações visualmente mais atrativas, nesta simples aplicação já podemos perceber alguns diferenciais visuais, como o foco diferenciado nos controles e até mesmo os ícones de algumas mensagens. Estes dois casos podem ser vistos na **Figura 15** que mostra a aplicação de exemplo em execução.

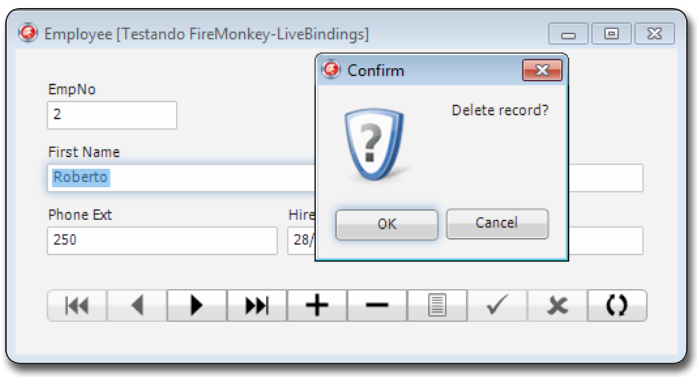


Figura 15. Aplicação de exemplo em execução

Conclusão

Com o lançamento de sua nova versão, a XE2, o Delphi apresentou novamente uma gama enorme de novidades. Seria inviável apresentar todas elas em um único artigo, dessa forma, este overview serviu então para explorar um pouco de seus novos e principais recursos. Apesar de superficial, com esta visão geral pudemos conferir as novidades em várias áreas diferentes, como aplicativos móveis, multi-plataforma, relatórios, estilos visuais, binding, Mac, 64 bits, além do tão falado FireMonkey. Em suma, teremos muitos assuntos interessantes a serem abordados e aprofundados em edições futuras desta revista, como também muito coisa nova para estudarmos e assim nos mantermos atualizados.

Com este novo release a Embarcadero prova mais uma vez que está investindo pesado no produto, procurando oferecer os recursos necessários para que o desenvolvedor Delphi se sinta plenamente atendido por seu IDE de trabalho. Prova disso é o suporte ao tão aguardado desenvolvimento para a plataforma 64 bits, além da produção de aplicações nativas para Mac (MacOS X e iOS). Além destes, outro recurso que causou certo frisson na comunidade foi o FireMonkey, o novo framework para desenvolvimento de aplicações ricas e de caráter multi-plataforma. Ricas pelo fato de apresentarem um visual espetacularmente atrativo, em vista do aproveitamento dos recursos de processamento gráfico da máquina.

Para finalizar, a dica que fica é para que se baixe esta nova versão do Delphi, disponibilizada gratuitamente na forma de avaliação no site da Embarcadero, e comece a explorar todas estas novidades apresentadas aqui. Dessa forma, você se manterá plenamente atualizado com sua ferramenta de trabalho.

Agradeço a todos e até a próxima.

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador Delphi há 5 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas além de participar do programa Beta Tester Delphi Prism, da Embarcadero.

Delphi XE2 Architect - Trial
https://downloads.embarcadero.com/free/delphi

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

DevMedia

DÊ UM SALTO EM CONHECIMENTO!

Acesse o maior portal para desenvolvedores da América Latina!

17 mil posts

280 mil cadastrados

5 milhões de page-views por mês

http://www.devmedia.com.br/

DevMedia

Como criar bots da inter expressões regulares em

Resumo DevMan

De que trata o artigo:

Este artigo ensina como criar um cliente telnet e também como utilizar expressões regulares e de modo geral como combinar estes conhecimentos para criar um bot que processa informações através da internet.

Em que situação o tema é útil:

Bots são muito úteis para realizar diversas tarefas na internet. Neste artigo utilizamos um bot que se conecta com o cliente telnet do Free Internet Chess Server para poder jogar xadrez através da internet utilizando uma interface mais agradável, por exemplo, uma interface gráfica. Similarmente é possível criar bots para automatizar qualquer tarefa envolvendo telnet ou de modo geral envolvendo comunicação através da internet e a análise de texto com expressões regulares.

Como criar bots da internet usando telnet e expressões regulares em Pascal:

LNet é uma das principais bibliotecas de sockets da linguagem Object Pascal. Ela fornece conexões não-bloqueantes o que facilita a escrita de programas, pois não é necessário criar linhas de execução separadas para tratar da comunicação por sockets. Esta biblioteca oferece suporte para os protocolos de envio de pacotes TCP e UDP e para os protocolos Telnet, HTTP, FTP, SSL e SMTP. Também há outras bibliotecas disponíveis para os desenvolvedores object Pascal, como Indy e Sinapse. A classe TRegExpr é o método mais difundido para se utilizar expressões regulares na linguagem de programação Object Pascal. Esta biblioteca funciona igualmente bem com o Free Pascal e com o Delphi e recentemente ela foi relicenciada na licença LGPL com exceção de ligamento estático, que é a licença utilizada pelo Free Pascal e pelo Lazarus. Com isto foi possível incluí-la no Free Pascal e esta biblioteca estará no Free Pascal 2.6. Esta biblioteca utiliza a sintaxe do Perl para expressões regulares que é uma sintaxe poderosa e flexível.

O nicho de criação de bots da internet é amplamente dominado por linguagens de script, principalmente Perl e, por isso, a maioria dos tutoriais disponíveis são para estas linguagens crípticas. Bots da internet são aplicativos que realizam automaticamente tarefas que normalmente deveriam ser feitas por uma pessoa, tarefas que envolvem a conexão com um servidor na internet. O que poucos sabem é que também há bibliotecas poderosas para Pascal que permitem a criação desse tipo de aplicativo, com a vantagem adicional da sintaxe do Pascal ajudar a criar aplicativos mais rapidamente, pois os erros são encontrados na compilação, ao contrário de linguagens com tipos dinâmicos onde os erros aparecem somente na execução. Neste caso em particular vamos estudar a criação de um bot que se conecta via Telnet.

Telnet é um protocolo que permite se comunicar com um servidor por uma interface do tipo terminal de texto, que é similar ao console que qualquer ambiente UNIX possui. A diferença é que os comandos não são os comandos do UNIX, mas sim o que o servidor fornece.

net usando telnet e Pascal

Felipe Monteiro de Carvalho

Essa interface foi desenvolvida para que pessoas possam se conectar ao servidor e executar comandos e ler dados, mas se tornou popular a criação de aplicativos que se comunicam com o servidor via Telnet e apresentam os dados para o usuário de uma forma mais amigável, por exemplo, através de uma interface gráfica com janelas.

Neste artigo estudaremos como utilizar a biblioteca de comunicação por rede lNet em conjunto com a biblioteca de expressões regulares do Sorokin para criar um aplicativo que se comunica via Telnet com o Free Internet Chess Server. O programa irá se comunicar com este servidor e apresentar os movimentos do xadrez através do programa fpChess.

Explorando o Telnet

Antes de começar, é indispensável saber como funciona o servidor ao qual vamos nos conectar. Para isso devemos utilizar um aplicativo para abrir uma conexão telnet com o servidor do Free Internet Chess Server. No Linux e no Mac OS X basta abrir o console e digitar "telnet freechess.org 5000". No Windows, o mesmo procedimento pode ser feito, bastando abrir o programa de terminal "cmd.exe".

Após fazer isto, logo irá se iniciar a conexão e seremos recebidos pelo servidor com um logo em arte ASCII, uma mensagem de boas vindas, com um prompt para inserir o nome de usuário que utilizaremos, como ilustra a **Figura 1**.

Após inserir o nome de usuário e pressionar "enter", aparece uma mensagem pedindo para confirmar o nome de usuário e daí é necessário pressionar "enter" novamente. Após esse segundo enter é que realmente nos logamos no servidor e podemos executar comandos.

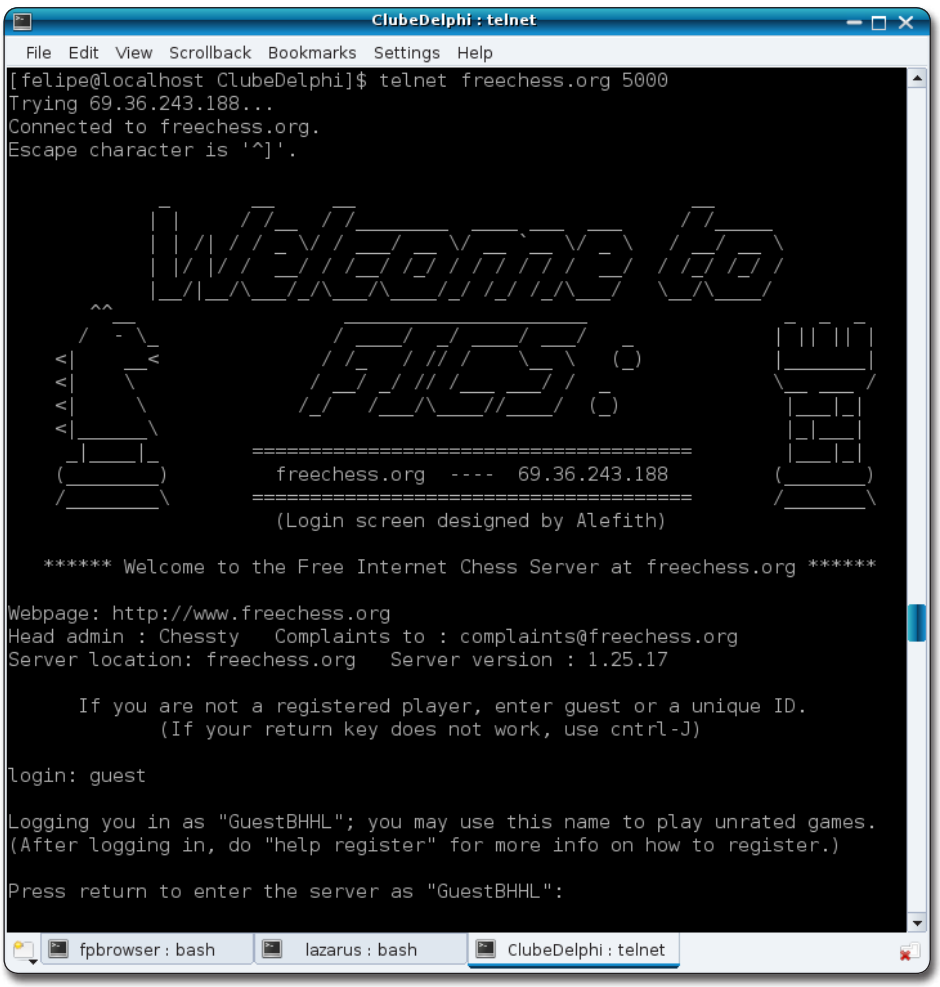


Figura 1. Tela de início do Free Internet Chess Server

Os comandos principais são: "set seek 0", que elimina um monte de mensagens de pessoas procurando parceiros para jogar xadrez que aparecem por padrão; "set style 11", que muda o estilo de jogo para o estilo número 11 (padrão) e ; "match outrousuario 60 White", que desafia um outro usuário a jogar uma partida de xadrez em que cada um terá 60 minutos de tempo e joga-se com as brancas.

Estes comandos são suficientes para iniciarmos o jogo. Na **Figura 2** está representada a tela inicial do FICS após o login.

É possível abrir duas sessões de telnet no mesmo computador e jogar contra si mesmo para aprender como o FICS funciona. O primeiro passo é se logar em cada terminal, como mostrado anteriormente e dar o comando "set seek 0" em cada terminal para remover as mensagens de pessoas procurando parceiros de xadrez. Depois basta usar "match outro usuario 60 White" para iniciar uma partida.

O outro usuário deverá responder com o comando "accept" para que a partida possa iniciar. Num dos terminais podemos dar o comando "set style 11" para ver o estilo mais adequado para o uso por um Bot, enquanto que outro pode ter a visão padrão, feita para humanos.

Para realizar um movimento basta escrever algo como "e2-e4", que significa que movemos a peca na posição E2 para a posição E4, e os dois jogadores receberão o novo estado do jogo após esse movimento, cada um no estilo que escolheu. Na **Listagem 1** podemos ver como se apresenta no terminal essa sequência de jogo no estilo 11.

Na **Listagem 2** podemos ver como se apresenta no terminal essa sequência de jogo no estilo padrão.

Se conectando ao Telnet com lNet

Há três principais bibliotecas de rede para utilizar em Pascal: Indy, Synapse e lNet. Indy é considerada de uso complexo, por isso iniciantes preferem Synapse ou lNet. A principal diferença entre essas duas bibliotecas é que Synapse opera com sockets síncronos, ou seja, ao esperar por um pacote ela trava a execução do programa até que ele chegue. Isto pode ser tanto uma grande facilidade como um grande problema. Por exemplo, programas visuais ficarão travados se funções da Synapse forem utilizadas na linha de execução principal. Para resolver este problema basta colocar as chamadas de rede numa linha de execução separada com ajuda da classe TThread, mas isto é um estorvo para programas simples.

Já a biblioteca lNet utiliza sockets assíncronos, os seja, ela não bloqueia a execução do programa. Os dados são recebidos em eventos ou então, é possível chamar uma função que retorna os dados que já chegaram imediatamente. Com qualquer das três bibliotecas é possível trabalhar com dados enviados pela internet, mas para este exemplo, escolhemos trabalhar com lNet, pois esta biblioteca fornece uma unidade chamada ltelnet que encapsula a comunicação Telnet de modo conveniente.

Basicamente o que o programa deverá fazer é simplesmente recriar os passos que um usuário normal faria ao se conectar ao servidor telnet. Abaixo podemos ver a estrutura principal do modulo do aplicativo fpchess e também o código para se conectar ao servidor. Não é importante se concentrar em como criar um módulo do fpchess, mas basta saber que é necessário criar uma classe derivada da classe TChessModule, como no exemplo da **Listagem 3**.

Adicionamos um campo TelnetComm do tipo TLTelnetClientEx e este será a classe principal que utilizaremos para se comunicar com o servidor via Telnet. Esta classe é uma extensão da classe disponível na biblioteca lNet, que é a TLTelnetClient e está disponível na unidade ltelnetex, que apresentaremos mais para frente. Essa classe TLTelnetClientEx adiciona suporte para utilizar expressões regulares junto com a conexão Telnet, de uma maneira inspirada pela biblioteca de rede do Perl.

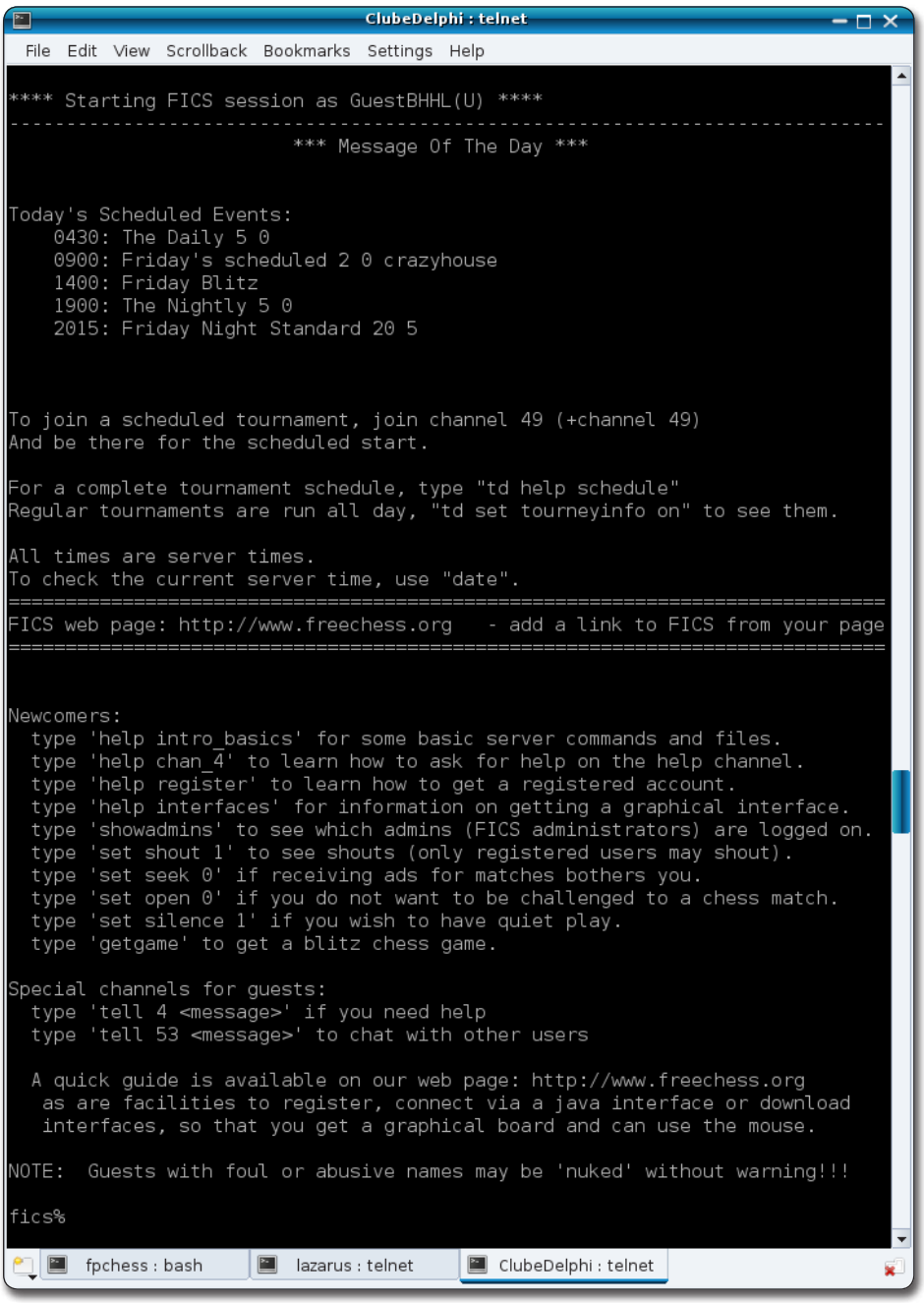


Figura 2. Tela do Free Internet Chess Server após o login

Listagem 1. Jogando xadrez no FICS com o estilo 11

```
fics% set seek 0
You will not see seek ads.
fics% set style 11
Style 11 set.
fics%
Challenge: felipemdc (----) felipemdcc (----) unrated blitz 2 12.
You can "accept" or "decline", or propose different parameters.
fics% accept
You accept the match offer from felipemdc.

Creating: felipemdcc (++++) felipemdc (++++) unrated blitz 2 12
{Game 219 (felipemdcc vs. felipemdc) Creating unrated blitz match.}

Game 219 (felipemdcc vs. felipemdc)

#@#219felipemdcc    :felipemdc    *RNBQKBNRPPPPPPPP
ppppppppprnbqkbnr001W39390012000120none   (0:00)@#@
fics% e2-e4

Game 219 (felipemdcc vs. felipemdc)

#@#219felipemdcc    :felipemdc    *RNBQKBNRPPPP PPP
Pppppppppprnbqkbnr001B39390012000120P/e2-e4(0:00)@#@
fics%
```

Listagem 2. Jogando xadrez no FICS com o estilo padrão

```
fics% set seek 0
You will not see seek ads.
fics% match felipemdcc
Issuing: felipemdc (----) felipemdcc (----) unrated blitz 2 12.
fics%
felipemdcc accepts the match offer.

Creating: felipemdcc (++++) felipemdc (++++) unrated blitz 2 12
{Game 219 (felipemdcc vs. felipemdc) Creating unrated blitz match.}

Game 219 (felipemdcc vs. felipemdc)

   ---------------------------------
 1 | R | N | B | K | Q | B | N | R |   Move # : 1 (White)
   |---+---+---+---+---+---+---+---|
 2 | P | P | P | P | P | P | P | P |
   |---+---+---+---+---+---+---+---|
 3 |   |   |   |   |   |   |   |   |
   |---+---+---+---+---+---+---+---|
 4 |   |   |   |   |   |   |   |   |   Black Clock : 2:00
   |---+---+---+---+---+---+---+---|
 5 |   |   |   |   |   |   |   |   |   White Clock : 2:00
   |---+---+---+---+---+---+---+---|
 6 |   |   |   |   |   |   |   |   |   Black Strength : 39
   |---+---+---+---+---+---+---+---|
 7 | *P| *P| *P| *P | *P| *P | *P| *P|   White Strength : 39
   |---+---+---+---+---+---+---+---|
 8 | *R|*N| *B | *K| *Q| *B| *N |*R|
   ---------------------------------
     h  g  f  e  d  c  b  a
fics%
Game 219 (felipemdcc vs. felipemdc)

   ---------------------------------
 1 | R | N | B | K | Q | B | N | R |   Move # : 1 (Black)
   |---+---+---+---+---+---+---+---|
 2 | P | P | P |   | P | P | P | P |   White Moves : 'e4  (0:00)'
   |---+---+---+---+---+---+---+---|
 3 |   |   |   |   |   |   |   |   |
   |---+---+---+---+---+---+---+---|
 4 |   |   |   | P |   |   |   |   |   Black Clock : 2:00
   |---+---+---+---+---+---+---+---|
 5 |   |   |   |   |   |   |   |   |   White Clock : 2:00
   |---+---+---+---+---+---+---+---|
 6 |   |   |   |   |   |   |   |   |   Black Strength : 39
   |---+---+---+---+---+---+---+---|
 7 | *P| *P| *P | *P| *P | *P| *P | *P|   White Strength : 39
   |---+---+---+---+---+---+---+---|
 8 | *R| *N| *B| *K| *Q| *B | *N| *R|
   ---------------------------------
     h  g  f  e  d  c  b  a
```

No método TFICSChessModule.Create vemos como criar uma instância dessa classe, como liberá-la no método TFICSChess Module.Destroy e, no método TFICSChessModule.Prepare ForGame pode-se ver como utilizá-la para iniciar uma conexão com um servidor Telnet.

O método para se conectar é bem fácil de ser reconhecido. Ele se chama Connect e aceita a URL do servidor e a porta dele, que é exatamente freechess.org e 5000, como ao utilizar o comando telnet no terminal. Mas algo que pode pegar um iniciante desprevenido aqui é que em lNet tudo é assíncrono, inclusive a conexão. Após chamar o método de conexão é necessário entrar num laço (loop) e esperar que a conexão realmente seja efetivada. Nesse loop devemos chamar TLTelnetClient.CallAction para processar qualquer dado que tenha chegado e a conexão estará realmente efetuada quando TLTelnetClient.Connected retornar True.

Após se conectar, devemos realizar os próximos passos para iniciar um jogo com um parceiro. Para isso, precisamos utilizar apenas dois métodos: TLTelnetClientEx.WaitFor e TLTelnetClient. SendMessage. O WaitFor utiliza expressões regulares para ler as mensagens que o servidor envia e ele irá esperar num loop até que o padrão procurado seja encontrado ou até que o tempo máximo de espera seja atingido. Esta função pode receber uma ou duas strings contendo expressões regulares e irá interromper seu laço (loop) quando qualquer uma das duas for encontrada.

As expressões regulares devem ser entradas em formato POSIX padrão e a biblioteca RegExpr.pas do Sorokin será utilizada para processar as expressões. No POSIX, um ponto representa um caracter qualquer e um asterisco representa zero ou mais ocorrências do item anterior. Assim, a expressão ".*login:.*", por exemplo, representa quaisquer caracteres, seguidos pela string "login:" seguidos por mais outros caracteres quaisquer. Note que as aspas não fazem parte da expressão.

Listagem 3. Declaração do módulo e conexão ao servidor telnet

```
TFICSChessModule = class(TChessModule)
// ...
public
  TelnetComm: TLTelnetClientEx;
  FICS_HOST: string;
  FICS_PORT: Integer;
  FICS_USER: string;
  FICS_PASSWORD: string;
  constructor Create(); override;
  destructor Destroy; override;
  procedure CreateUserInterface(); override;
  procedure ShowUserInterface(AParent: TWinControl); override;
  procedure HideUserInterface(); override;
  procedure FreeUserInterface(); override;
  procedure PrepareForGame(); override;
  function GetSecondPlayerName(): string; override;
  procedure HandleOnMove(AFrom, ATo: TPoint); override;
  procedure HandleOnTimer(); override;
  procedure HandleOnDebugOut(AStr: string);
end;

const
  FICS_LineEnding = #10;
  OPEN_TIMEOUT = 1000000;
  PROTECT_LOGOUT_FREQ = 45 * 60 * 1000; // Frequency to issue commands to
  avoid disconnection, in miliseconds

implementation

{ TFICSChessModule }

constructor TFICSChessModule.Create;
begin
  inherited Create;

  TelnetComm := TLTelnetClientEx.Create(nil);
  TelnetComm.OnDebugOut := @HandleOnDebugOut;

  Name := 'mod_fics.pas';
  SelectionDescription := 'Play online - Free Internet Chess Server';
  PlayingDescription := 'Playing online - Free Internet Chess Server';
  Kind := cmkInternet;

  FICS_HOST := 'freechess.org';
  FICS_PORT := 5000;
  FICS_USER := 'FPChess';
  FICS_PASSWORD := '';
end;

destructor TFICSChessModule.Destroy;
begin
  TelnetComm.Free;

  inherited Destroy;
end;

procedure TFICSChessModule.PrepareForGame;
var
  lResult, WaitTerminated: Boolean;
  lMsg: string;
begin
  FICS_USER := vChessGame.PlayerName;
  FICS_PASSWORD := editPassword.Text;

  ChessModuleDebugLn('[TFICSChessModule.PrepareForGame]');

  //  Opening telnet connection. This is what happens when you issue telnet freechess.
  //  org 5000.

  lResult := TelnetComm.Connect(FICS_HOST, FICS_PORT);
  if not lResult then
  begin
    ChessModuleDebugLn('Failed to connect to FICS');
    Exit;
  end;

  repeat
    TelnetComm.CallAction; // repeat this to get info
    Application.ProcessMessages;
    Sleep(10);
  until TelnetComm.Connected; // wait until timeout or we actually connected
```

Já o método SendMessage envia uma mensagem ao servidor. Devemos incluir o enter na mensagem, e ele é representado pela constante FICS_LineEnding que já foi declarada anteriormente na **Listagem 3.** O Free Pascal possuí a constante LineEnding, que representa o fim de linha da plataforma onde o programa está executando, mas não convêm utilizá-la aqui, pois não convêm que o fim de linha varie de acordo com a plataforma onde o programa roda, já que o servidor é sempre o mesmo e independe da plataforma do programa.

A sequência de eventos após a conexão pode ser vista na **Listagem 4**. Primeiro esperamos pela mensagem pedindo nosso login e daí enviamos para o servidor, seguido de um "enter". Após a mensagem de início da sessão, aparecerá uma linha de comando. Isso significa que podemos escrever comandos para que o servidor os responda. Deste ponto em diante poderemos sempre esperara pela linha de comando representada por "fics%" para esperar que o FICS processe um comando.

O primeiro comando a ser enviado será o "set seek 0" para evitar o tumulto no console que poderia vir de outra forma. Depois utilizamos "set style 11" para mudar o estilo de representação do tabuleiro e dos movimentos. A partir daqui os eventos diferem para o jogador que espera por uma conexão e para aquele que se conecta. O jogador que se conecta envia o comando "match", enquanto que o outro jogador responde com "accept". Não importa muito exatamente os detalhes de como o FICS opera aqui, pois o mais importante é aprender a utilizar a classe TLTelnetClientEx para criar outros aplicativos e, para isto, basta utilizar um processo semelhante, envolvendo chamadas a WaitFor e chamadas a SendMessage.

A classe TLTelnetClientEx e Expressões Regulares

Se por um lado não é importante saber exatamente os detalhes de como o servidor FICS funciona, por outro é indispensável saber como utilizar expressões regulares para poder criar seus próprios bots. Quem desejar saber como se conectar ao FICS em mais detalhes poderá ler a unidade mod_fics.pas do fpChess. A unidade inteira ltelnetex pode ser vista na **Listagem 5** e ela é muito importante para utilizar Telnet de um modo parecido com o que se faz em Perl pois ela adiciona suporte a função WaitFor com expressões regulares.

A biblioteca de expressões regulares utilizada é a regexpr.pas do Sorokin, que é mais popular. Esta biblioteca é utilizada pelo Lazarus e também foi adicionada no Free Pascal 2.6.0 que está prestes a ser lançado.

Para manter a compatibilidade com as versões, tanto novas quanto antigas do Free Pascal e do Delphi, podemos simplesmente copiar a unidade regexpr.pas do Free Pascal e renomeá-la para utiliza-la em nosso projeto, o que foi feito para a unidade ltelnetex no fpChess. A unidade de expressões regulares foi renomeada para sorokinregexpr, para que funcione com versões antigas do Free Pascal e não dê conflitos quando a unidade aparecer no próximo Free Pascal.

Conforme a **Listagem 5,** o primeiro passo é criar uma instância dessa classe com o construtor TRegExpr.Create. Depois devemos colocar a expressão regular na forma de uma string na propriedade TRegExpr.Expression. O texto a ser inspecionado para verificar se há uma expressão regular nele deve ser colocado no único parâmetro do método TRegExpr.Exec. Este método ira retornar True se a expressão regular for encontrada no texto ou False se ela não for.

Um exemplo mais compacto de como utilizar o TRegExpr pode ser visto a seguir. Esse exemplo irá verificar que a expressão ".*Monteiro.*" que está no texto "Felipe Monteiro de Carvalho". Como .* representa zero ou mais ocorrências de qualquer caráctер, o resultado será positivo:

```
var
 RegexObj: TRegExpr;
begin
 RegexObj := TRegExpr.Create;
 RegexObj.Expression := '.*Monteiro.*';
 Result := RegexObj.Exec('Felipe Monteiro de Carvalho');
 RegexObj.Free;
```

Há vários padrões para escrever expressões regulares, sendo que há dois padrões: POSIX e o padrão criado para a linguagem de programação Perl. Esta linguagem é caracterizada por sua sintaxe críptica, mas ela possuí um excelente suporte a expressões regulares, criando uma sintaxe melhor que a dos padrões POSIX e implementando uma grande variedade de funções. Hoje, praticamente todas as maiores implementações de expressões regulares se baseiam na sintaxe das expressões regulares do Perl e o TRegExpr não é exceção. O TRegExpr suporta parte da sintaxe do Perl (a sintaxe das expressões regulares do Perl, que não deve ser confundida com a sintaxe da linguagem Perl), faltando suporte para algumas capacidades mais avançadas e menos utilizadas.

O padrão mais simples já foi mostrado nos exemplos anteriores,

Listagem 4. Se conectando a outro jogador no Free Internet Chess Server

```
 TelnetComm.WaitFor(
  '.*login:.*',
  '.*username:.*',
  OPEN_TIMEOUT);
 // ... and we send our username once prompted.
 ChessModuleDebugLn('Sending: ' + FICS_USER);
 TelnetComm.SendMessage(FICS_USER + FICS_LineEnding);
 // Now we read obtained lines scanning for some patterns.
 TelnetComm.WaitFor(
  '.*Press return to enter.*',
  '',
  OPEN_TIMEOUT);
 TelnetComm.SendMessage(FICS_LineEnding);
 TelnetComm.WaitFor(
  '.*Starting FICS session as.*',
  '',
  OPEN_TIMEOUT);
end;
// Remove those annoying message of people seeking chess adversaries
TelnetComm.WaitFor(
 '.*fics%.*',
 '',
 OPEN_TIMEOUT);
ChessModuleDebugLn('Sending: set seek 0');
TelnetComm.SendMessage('set seek 0' + FICS_LineEnding);
// Set the style
TelnetComm.WaitFor(
 '.*fics%.*',
 '',
 OPEN_TIMEOUT);
ChessModuleDebugLn('Sending: set style 11');
TelnetComm.SendMessage('set style 11' + FICS_LineEnding);
// Wait for a match
if radioConnectionType.ItemIndex = 0 then
begin
 vChessGame.FirstPlayerIsWhite := False;
 // Challenge: GuestZMYL (----) fpchess (----) unrated blitz 2 12.
 // You can "accept" or "decline", or propose different parameters.
 TelnetComm.WaitFor(
  '.*You can "accept" or "decline", or propose different parameters*',
  '',
  OPEN_TIMEOUT);
 ChessModuleDebugLn('Sending: accept');
 TelnetComm.SendMessage('accept' + FICS_LineEnding);
 // You accept the match offer from GuestZMYL.
 TelnetComm.WaitFor(
  '.*You accept the match offer from*',
  '',
  OPEN_TIMEOUT);
end
// Challenge a partner
else
begin
 TelnetComm.WaitFor(
  '.*fics%.*',
  '',
  OPEN_TIMEOUT);
 lMsg := 'match ' + editSecondPlayerName.Text + ' 60 White';
 ChessModuleDebugLn('Sending: ' + lMsg);
 TelnetComm.SendMessage(lMsg + FICS_LineEnding);
 // fpchess accepts the match offer.
 TelnetComm.WaitFor(
  '.*accepts the match offer*',
  '',
  OPEN_TIMEOUT);
end;
end;
```

Listagem 5. A unidade ltelnetex

```
unit ltelnetex;
{$mode objfpc}{$H+}
interface
uses
  Classes, SysUtils,
  lTelnet, lnet,
  sorokinregexpr,
  Forms;
type
  TLTelnetDebugOutProc = procedure (AStr: string) of object;
  { TLTelnetClientEx }
  TLTelnetClientEx = class(TLTelnetClient)
  private
    FOnDebugOut: TLTelnetDebugOutProc;
    RegexObj: TRegExpr;
  public
    LastMsg: string;
    constructor Create(aOwner: TComponent); override;
    destructor Destroy; override;
    function WaitFor(FirstMatch, SecondMatch: string; ATimeOut: Cardinal): Integer;
    property OnDebugOut: TLTelnetDebugOutProc read FOnDebugOut write FOnDe-
bugOut;
  end;
implementation
{ TLTelnetClientEx }
constructor TLTelnetClientEx.Create(aOwner: TComponent);
begin
  inherited Create(aOwner);
  RegexObj := TRegExpr.Create;
end;
destructor TLTelnetClientEx.Destroy;
begin
  RegexObj.Free;
  inherited Destroy;
end;
{ Returns a zero-based index to which regular expression was found
  or -1 if none was found
}
function TLTelnetClientEx.WaitFor(FirstMatch, SecondMatch: string;
  ATimeOut: Cardinal): Integer;
var
  lMsg: string;
  lRemaining: Integer;
begin
  lRemaining := ATimeOut;
  Result := -1;
  if (FirstMatch = '') and (SecondMatch = '') then Exit;
  repeat
    if GetMessage(lMsg) > 0 then
      if Assigned(OnDebugOut) then OnDebugOut(lMsg);
    LastMsg := lMsg;
    if FirstMatch <> '' then
    begin
      RegexObj.Expression := FirstMatch;
      if RegexObj.Exec(lMsg) then Exit(0);
    end;
    if SecondMatch <> '' then
    begin
      RegexObj.Expression := SecondMatch;
      if RegexObj.Exec(lMsg) then Exit(1);
    end;
    CallAction; // don't forget to make the clock tick :)
    if lRemaining > 0 then // Don't sleep if the routine was called with ATimeOut=0
    begin
      Application.ProcessMessages;
      Sleep(100);
    end;
    Dec(lRemaining, 100);
  until lRemaining <= 0;
end;
end.
```

trata-se do elemento "." que representa qualquer carácter e do iterador "*" que representa zero ou mais ocorrências do elemento anterior. É possível combinar ambos, fazendo ".*" que representa zero ou mais ocorrências de qualquer carácter. Outros caracteres inseridos serão tradados e com eles é possível agrupar elementos com parênteses para utilizar o iterador "*". Um exemplo é a expressão "(ab)*" que bate com os seguintes textos: "", "ab", "abab", "abababab", etc. Pode-se ver que não é possível escrever diretamente os caracteres com significados especiais. Para escrever algum deles, é necessário escrever uma combinação de uma barra seguida do carácter, por exemplo, "\.". Para obter o carácter de uma barra é necessário escrever duas barras, por exemplo, "\\". Esta combinação é chamada de sequência de escape.

Na **Tabela 1** podemos ver várias sequências de escape suportadas pelo TRegExpr.

Além de obter a resposta indicando se uma expressão regular se encaixa ou não no texto fornecido, também é possível obter qual foi exatamente o subtexto do texto que se encaixou na expressão. Isto é feito com a ajuda da propriedade do tipo somente-leitura Match. Esta propriedade também pode passar subtextos secundários que poderiam ser aceitos, mas seu uso mais comum será utilizando o valor zero para o seu único parâmetro, a fim de identificar que o subtexto primário deverá ser retornado.

Note que deve-se verificar se o método TRegExpr.Exec retornou True e somente ler Match[0] em caso positivo. O exemplo a seguir mostra como usar TRegExpr para ler qual o subtexto que irá ser retornado ao passar a expressão regular "(1)?2(3)?" no texto "123". O resultado será Match[0]="123":

```
var
  RegexObj: TRegExpr;
  i: Integer;
begin
  RegexObj := TRegExpr.Create;
  RegexObj.Expression := '(1)?2(3)?';
  if RegexObj.Exec('123') then
    ResultStr := RegexObj.Match[0];
  RegexObj.Free;
```

Além de especificar caracteres, também é possível especificar iteradores. O mais simples deles já foi explicado, que é o "*", mas há muitos outros iteradores. Iteradores indicam quantas ocorrências de uma expressão devem ser aceitas. Nesse momento é importante distinguir os iteradores gananciosos de iteradores não-gananciosos (em inglês se utiliza os termos "greedy" e "non-greedy"). Iteradores gananciosos irão obter o máximo possível de elementos que satisfazem a condição enquanto que, iteradores não-gananciosos irão retornar o mínimo possível de elementos que satisfazem a condição.

Expressão	Descrição	Exemplo
\xnn	Um byte com o código hexadecimal nn	/x61 que bate com o carácter #$61, que representa "a" em ASCII e UTF-8
\x{nnnn}	Um ou mais bytes representados em hexadecimal. É útil para representar caracteres Unicode	/x{0061} que é a sequência de bytes $00 $16, que representa "a" em UTF-16
\t	Tabulação, #$09 em ASCII e UTF-8	
\n	Nova linha, #10 em ASCII e UTF-8	
\r	Nova linha, #13 em ASCII e UTF-8	
\\	Representa "\"	
\^	Representa "^"	
\-	Representa "-"	
\+	Representa "+"	
[abc]	Aceita "a", "b" ou "c"	
[^abc]	Aceita um carácter qualquer exceto "a", "b" ou "c"	
[a-c]	Aceita "a", "b" ou "c"	
^	Início da linha	Por exemplo "^Bom dia" aceita a expressão "Bom dia", mas somente se ela aparecer no início de uma linha
$	Fim da linha	"Tchau$" aceita "Tchau", mas somente no final de uma linha
\A	Início do texto	"\ABom dia" aceita "Bom dia", mas somente no início do texto
\Z	Fim do texto	"Tchau\Z" aceita "Tchau", mas somente no final do texto
.	Um carácter qualquer	
\w	Um carácter alfanumérico (incluindo "_")	
\W	Um carácter não alfanumérico	
\d	Um carácter numérico	
\D	Um carácter não numérico	
\s	Um espaço qualquer, que pode ser espaço, tabulação ou fim de linha (o mesmo que [\t\n\r\f])	
\S	Qualquer carácter exceto um espaço do tipo \s	
\b	Representa uma separação entre palavras, que se inicia após um \w onde se inicia uma sequência de elementos \W até o próximo \w	

Tabela 1. Sequências de escape suportadas pelo TRegExpr

O iterador mais flexível de todos é o "{n,m}" que define que a expressão deverá aparecer no mínimo n vezes, mas no máximo m vezes. Também há formas simplificadas, como {n} que é equivalente a {n,n} e indica que a expressão deverá aparecer exatamente n vezes. Outra forma de escrever é {n,} que indica que não há limite superior, assim, a expressão deverá aparecer n ou mais vezes. Quando o limite inferior é diferente do superior é possível estabelecer o conceito de iterador ganancioso ou não-ganancioso. Por exemplo, com a expressão regular "a{2,3}" e o texto "aaa" o resultado de Match[0] será "aaa", mas se utilizarmos o iterador não-ganancioso, teremos a expressão regular "a{2,3}?" e o resultado de Match[0] será "aa". Não há limite para os números n ou m, mas números exageradamente

Expressão	Descrição
*	Zero ou mais ocorrências ganancioso, similar a {0,}
+	Uma ou mais ocorrências ganancioso, similar a {1,}
?	Zero ou uma ocorrência ganancioso, similar a {0,1}
{n}	Exatamente n ocorrências ganancioso
{n,}	Pelo menos n ocorrências ganancioso
{n,m}	Pelo menos n ocorrências, mas não mais que m, ganancioso
*?	Zero ou mais ocorrências não-ganancioso, similar a {0,}?
+?	Uma ou mais ocorrências não-ganancioso, similar a {1,}?
??	Zero ou uma ocorrência não-ganancioso, similar a {0,1}?
{n}?	Exatamente n ocorrências não-ganancioso
{n,}?	Pelo menos n ocorrências não-ganancioso
{n,m}?	Pelo menos n ocorrências, mas não mais que m, não-ganancioso

Tabela 2. Principais iteradores

grandes poderão tornar a execução da expressão regular lenta e consumir mais memória.

Os principais iteradores estão explicados na **Tabela 2** e combinando os caracteres, caracteres de escape e iteradores é possível explorar boa parte do poder das expressões regulares.

Conclusão

Neste artigo foi mostrado como criar controles um cliente telnet utilizando as classes TTelnetClient e TTelnetClientEx. Também foi mostrado como utilizar um cliente telnet para criar um bot da internet, que é um aplicativo que realiza atividades automaticamente através da internet. Além disso também foi explicado como utilizar expressões regulares através da biblioteca TRegExpr.

Expressões regulares são um tema muito amplo e somente a sintaxe mais básica e simples foi explicada, que é suficiente para começar a utilizar expressões regulares para analisar textos. Com estes conhecimentos já é possível começar a criar seus próprios bots e a explorar mais profundamente a biblioteca lNet e as expressões regulares.

Felipe Monteiro de Carvalho
felipemonteiro.carvalho@gmail.com
Engenheiro elétrico formado pela Escola Politécnica da USP. É desenvolvedor de diversos projetos de software livre, entre eles Lazarus, Free Pascal e o Virtual Magnifying Glass. Trabalha atualmente na Opera Software em Wrocław - Polônia na área de qualidade de software.

Documentação do lNet
http://wiki.lazarus.freepascal.org/lNet

Sintaxe do TRegExpr
http://regexpstudio.com/TRegExpr/Help/regexp_syntax.html

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Seu **conhecimento vale muito** na DevMedia

Publique artigos e vídeos gratuitamente no site e participe do programa **PayContent.**

Sua participação pode ser premiada: quanto mais você postar, mais chances de ganhar.

Para saber mais acesse:
www.devmedia.com.br/paycontent

DevMedia

Metodologias Ágeis

Um estudo sobre XP e Scrum

Em desenvolvimento de software, o termo Ágil significa, basicamente, a busca em simplificar as coisas, tanto em nível de projeto quanto de equipe. A parte de projeto pode ser traduzida em reduzir ao máximo o nível de complexidade de planejamento ao mesmo tempo em que todos os esforços são concentrados em priorizar aquilo que agrega maior valor para o cliente. Já em termos de equipe é a busca pelo estabelecimento da participação e colaboração necessária entre os envolvidos no projeto. Em busca dessa "agilidade", surgiram inúmeras técnicas, metodologias e melhores práticas, conforme será visto no decorrer do artigo.

História

Historicamente falando, décadas atrás, o Sistema Toyota de Produção ficou conhecido no meio industrial por ser uma solução, até então inovadora. Esta solução foi desenvolvida pela Toyota, renomada fábrica japonesa de automóveis, na época do pós-guerra (Segunda Guerra Mundial), onde a indústria japonesa apresentava uma produtividade muito baixa, comparado ao hemisfério ocidental, justificada por uma alastrante falta de recursos. Este fato, naturalmente impedia essas indústrias de adotarem um modelo de produção em massa.

O Sistema Toyota de Produção, também conhecido como Lean Manufacturing (Produção Enxuta), provia à fábrica uma capacidade de produção de melhor qualidade aliado a um menor custo de produção, o que o tornava muito superior aos sistemas de produção mais tradicionais, como Taylor e Ford. O segredo desta inovação girava em torno do termo japonês Kaizen que representa a dedicação pela melhoria contínua em todos os aspectos da vida. Passando isso para a parte de produção, tem-se então uma força motriz que se adapta ao processo de fabricação, aos clientes e às exigências do mercado, conforme os seguintes itens:

- Realização de pequenas tarefas em busca de uma melhora gradativa a cada dia;
- Definição de padrões e a busca por atingi-los numa escala crescente (padrões cada vez mais elevados);
- Melhora contínua em todas as áreas e em todos os níveis.

Além disso, a Toyota concentrou-se em eliminar qualquer tipo de obstáculo fora do projeto, permitindo uma fluidez maior em suas tarefas. Foi adotado também o conceito Just-in-Time, que visava produzir somente o necessário para atender às necessidades do momento, evitando assim filas de produção desnecessárias.

Resumo DevMan

De que trata o artigo:

O presente artigo tem como premissa principal apresentar um estudo abrangente sobre metodologias ágeis de software, expondo um pouco do seu histórico geral e de suas características principais. Serão feitas também menções sobre a empregabilidade de metodologias específicas. Como são várias as opções ágeis disponíveis, o enfoque será dado a duas das mais populares: eXtremme Programming (XP) e Scrum.

Em que situação o tema é útil:

Na perspectiva mais tradicional do desenvolvimento de software, o objetivo principal a ser atingido é o sucesso do projeto em si, enquanto que numa perspectiva ágil, este objetivo é substituído pela satisfação do cliente. Fica perceptível a mudança de paradigma entre uma abordagem e outra. Na prática, isto resulta numa nova forma de pensar e trabalhar, ou seja, uma nova filosofia. Enquanto que na metodologia tradicional há uma linha de processos bem definidos e formais, numa metodologia ágil, existe uma maior flexibilidade de suas práticas de acordo com o projeto.

Metodologias Ágeis:

Em termos gerais, duas das características comuns que definem uma metodologia ágil são o seu processo de desenvolvimento marcado por iterações que, num curto espaço de tempo, produz algo de valor ao cliente e o enfoque na comunicação interpessoal, deixando qualquer tipo de documentação em segundo plano. Em vista do alto grau de usabilidade, duas metodologias ganham destaque e serão as abordadas neste artigo: eXtreme Programming (XP) e Scrum. Com o intuito de apresentá-las, serão abordados pontos como a história da melhoria do processo de desenvolvimento e a definição geral que caracteriza um método como sendo ágil. Só então serão apresentadas as duas metodologias citadas, massificando a teoria sobre suas principais características e empregabilidade.

Todo o poder da produção enxuta está na forma de como ela é aplicada em uma organização e, mais que isso, na mudança de cultura ocasionada dentro da empresa. Algumas das melhores práticas deste tipo de produção são:

- Previsão de planejamento e pensamento;
- Padronização de processos;
- Melhoria contínua (teoria Kaizen);
- Trabalho e colaboração em equipe em detrimento à divisão de trabalho convencional;

Fabrício Hissao Kawata

• Descentralização de responsabilidades;
• Solução de erros através da busca pela raiz da causa;
• Foco tão somente no que pode agregar mais valor ao cliente;
• Manter-se flexível para eventuais requisitos imprevistos do cliente;
• Manutenção de processos simples.

Diante do que foi dito, este tipo de produção era até então voltada para o ramo industrial de manufatura, mas logo começou a ser também aproveitado na indústria de TI, ou seja, no desenvolvimento de software. Segundo Jeff Sutherland, criador do Scrum (metodologia ágil que será vista a seguir), todos os métodos ágeis para software podem ser considerados aplicações da teoria Lean voltada para a área.

Lean Software Development

De forma evolutiva, baseado nas melhores práticas da produção enxuta (lean) deu-se início ao desenvolvimento de software Lean, ou Lean Software Development. Esta metodologia apresenta como principais características:

• Foco na aprendizagem: diferente de um planejamento exacerbado, nesta metodologia o aprendizado é um item essencial. O uso de protótipos de projetos que visam ser ágeis o suficiente é uma boa alternativa. Com isso, é adquirida uma experiência significativa em termos de estimativas de projeto e atenção às necessidades reais do cliente, que resulta numa melhora dos padrões existentes. Este último item, aliás, está atrelado à melhoria constante, outro item importante. Esta melhoria pode ser representada pela busca progressiva da correção de falhas e erros no contexto do projeto, procurando saná-los a partir de uma análise mais profunda de suas causas, o que previne para que os mesmos não aconteçam novamente num futuro;

• Eliminação de resíduos: em desenvolvimento de software o termo "resíduo" é tudo que foge do escopo principal do projeto e consequentemente não apresenta valor real para o cliente naquele momento. Como um primeiro exemplo, pode-se citar a realização de uma atividade fora de foco atrelada a uma tecnologia não relacionada. Isso pode ser traduzido num caso cotidiano, onde o desenvolvedor envolvido no projeto, por um anseio pessoal, faz a aplicação de uma tecnologia totalmente nova ao projeto e que não agrega valor algum ao mesmo. Pelo contrário, dá margem para o surgimento de futuros novos bugs, além de uma manutenção mais custosa. Um design de projeto excessivamente complexo para o momento, onde o mesmo não poderá ser devidamente implementado dentro do prazo atual também pode ser considerado desperdício. Da mesma forma que o apego às previsões de aspectos futuros do projeto, além de correrem o risco eminente de mudanças, acabam por atrasar o processo atual que é o que deverá ser entregue no prazo;

• Construção com qualidade: em desenvolvimento de software, a palavra qualidade pode ser interpretada de maneiras diferentes, de acordo com a parte interessada. Justificando, para um desenvolvedor, a qualidade está relacionada a um código livre de erros e que, consequentemente, gera um produto final que atenda o cliente. Já para o cliente, a qualidade de um software está ligada a ter um produto que atenda suas expectativas, agregando valor ao seu negócio. Dessa forma, para a equipe do projeto, uma construção de qualidade é a combinação entre uma integração contínua de alterações de código com a flexibilidade em absorver as exigências de cada cliente, a fim de estabelecer os requisitos e o que agrega valor a cada individualidade;

• Adiar compromissos: quebrar o planejamento de um software em pequenos processos incrementais é uma abordagem mais próxima da realidade em comparação a um planejamento único para o projeto inteiro. Esta divisão em pequenos processos exige a quebra da dependência entre os componentes envolvidos, permitindo uma progressão independente, permitindo que a arquitetura geral do sistema seja tolerante a mudanças, isto porque cada membro da equipe será responsável por uma partícula do todo, que compõe o que precisa ser feito. Em outras palavras, que possibilite a adição de novos recursos a qualquer momento, ao mesmo tempo em que a equipe seja capaz de absorver tais mudanças, tratando-as como parte da estratégia para atingir um objetivo. Além disso, o projeto precisa ser capaz de explorar opções variadas para determinado problema. No entanto, a decisão concreta por uma escolha deve ser adiada ao extremo, com o intuito de que seja adquirida a devida experiência e conhecimento no decorrer do processo;

• Entrega rápida: a fim de se evitar o aglomerado de pilhas de trabalhos de uma equipe é utilizada uma abordagem onde menos coisas são feita durante um processo, visando aumentar o tempo de resposta. Em outras palavras, se estabelece um ritmo mais confiável de produção, limitando a carga de trabalho para uma capacidade mais realista, visando uma entrega rápida aliada à alta qualidade e menor custo. Neste tipo de abordagem, ao se trabalhar com iterações menores durante o processo, melhorias são disponibilizadas mais rapidamente ao cliente, ou seja, ele sempre terá algo palpável de valor à sua disposição;

• Respeito entre as pessoas: o termo respeito em Lean, visa incentivar a auto-responsabilidade e autonomia das equipes.

Uma equipe motivada, trabalhando em conjunto em busca de um mesmo objetivo, proporciona uma boa vantagem competitiva. Aliado a isso, está a presença de uma liderança que consegue se impor de forma eficaz, estimulando o orgulho e compromisso individual, através de respeito e confiança. Todo este clima faz com que todo o talento e criatividade existente aflorem mais facilmente. Além disso, é definido que uma decisão pode ser mais bem feita por uma pessoa que detenha um alto conhecimento técnico sobre uma situação. Assim sendo, as responsabilidades e tomadas de decisões são também delegadas às equipes, que são quem diretamente desenvolvem o produto de valor real ao cliente, apoiado nos detalhes técnicos para a elaboração do software;

• Otimização do todo: a otimização do todo se apoia na ideia do Kaizen, que prega que nada nem ninguém é perfeito. A melhoria contínua se dá em pequenos passos e uma iteração evolutiva pode ser o caminho ideal para a perfeição. Dessa forma, esta otimização está estritamente ligada à revisão contínua de dois itens fundamentais no projeto: o produto final e seu processo de desenvolvimento. Com isso, espera-se conseguir uma visão de alto nível do sistema como um todo e dos requisitos do cliente. Oportunidade e tecnologia é a combinação para atender todas as necessidades do cliente. O resultado desta revisão deve quantificar, entre outras coisas, itens como a satisfação do cliente em relação ao produto final e o tempo medido de desenvolvimento gasto.

Manifesto Ágil

Até meados de 2000, as ideias sobre desenvolvimento ágil eram bastante segmentadas e dispersas, o que contribuía para que o tema não fosse tão popular, flexível e mais bem sucedido quanto deveria. No entanto, este cenário começou a mudar no ano de 2001, quando dezessete profissionais da comunidade de software se reuniram e assinaram uma declaração, que ficou conhecida como Manifesto Ágil. Ela regia, de forma geral, todas as diferentes regras e técnicas envolvidas nos mais variados métodos ágeis existentes, a fim de declarar os melhores modos para se desenvolver software. Entre os signatários estavam nomes como Kent Beck, Brian Marick e Ken Schwaber, que estabeleceram os pilares do manifesto, fundamentados basicamente na flexibilidade e na colaboração, numa escala de relevância maior à rigidez de processos, do modelo clássico. Estes pilares podem ser resumidos em quatro valores fundamentais:

• Priorizar os indivíduos e suas interações acima de procedimentos, processos e ferramentas;

• Apresentar um software funcionando, ao invés de uma documentação abrangente;

• Preferência por um cliente colaborativo a uma negociação de contratos;

• Possuir uma capacidade de resposta a modificações, não ficando preso a um plano pré-estabelecido.

Metodologia Ágeis

O Manifesto Ágil, através de seus pilares, fundamentaram o desenvolvimento ágil de software, caracterizando um processo moderno que identifica a agilidade, voltando o foco do desenvolvimento para as eventuais modificações que possam ocorrer e a capacidade da equipe em responder a elas. Essa mesma filosofia ágil deu origem a várias metodologias ágeis, surgidas antes mesmo do manifesto, tais como Extreme Programming (XP), Feature Driven Development (FDD), Test Driven Development (TDD), Scrum entre outras. Neste cenário, duas se tornaram bastante populares: Scrum e XP.

O desenvolvimento ágil por si só estabelece alguns princípios básicos que caracterizam a valorização de alguns aspectos como:

• Simplicidade;

• Rápida adaptação às eventuais mudanças que possam ocorrer durante um processo;

• Entrega contínua de requisitos funcionais e de valor ao cliente, mantendo-o satisfeito;

• Foco na satisfação do cliente ao invés da busca tão somente pelo sucesso do projeto;

• Cooperação entre as pessoas envolvidas num projeto (cliente, desenvolvedores e demais interessados que possam existir);

• Foco na manutenção da motivação da equipe;

• Indivíduos e interações sobre processos e ferramentas;

• Preferência por um software funcional a uma documentação extensa;

• Cliente colaborativo.

Comparação com outros métodos

Em virtude da comparação inevitável que se faz entre uma metodologia ágil com outros tipos de metodologias, torna-se essencial, para um melhor entendimento, a exposição de algumas destas outras opções.

Modelo em cascata

O modelo de desenvolvimento em cascata é um modelo de desenvolvimento de software de caráter sequencial, cuja fluidez se dá como num movimento em cascata (para frente e em etapas). Sua ênfase gira em torno de um planejamento bem delineado através dos seguintes processos: requisitos (levantamento e análise), projeto, implementação (codificação), integração, testes (depuração), instalação e por fim a manutenção do software. Todos estes passos englobam o ciclo de vida do projeto, que podem durar longos meses para sua conclusão.

Devido à extensão do projeto, as dificuldades de integração e testes podem ser as causas substanciais de um possível insucesso. Este modelo também é considerado o mais tradicional, dentre os modelos de desenvolvimento. Para um melhor entendimento, a **Figura 1** exibe um esboço gráfico do modelo em cascata, destacando o fluxo de seus processos.

Conforme o exemplo apresentado na figura, na prática teríamos a seguinte situação: a etapa inicial localizada no topo do fluxo, denominada Requisitos, é o momento direcionado às tratativas relacionadas aos requisitos do software. Somente ao término desta etapa é que se dá início ao processo seguinte, denominado Projeto,

onde será realizada a elaboração do projeto do software a ser construído.

Novamente, somente de posse do projeto por completo é que se inicia o processo de implementação, onde os desenvolvedores efetivamente entram em ação, escrevendo código. Na etapa seguinte ocorre a integração dos componentes produzidos pela equipe de desenvolvimento, a fim de produzir um único produto final (o software propriamente dito).

Se e somente se o software estiver devidamente "montado" é que se inicia a fase de testes do mesmo, com a intenção de remover qualquer problema introduzido em etapas anteriores. Com mais esta etapa concluída, finalmente o software estará apto a ser instalado no cliente e, consequentemente, atingir o seu último processo, denominado Manutenção. Neste, o foco está na manutenção do software em produção, que caracteriza a introdução de novas funcionalidades e a remoção de possíveis erros que venham a ocorrer.

Modelo Balbúrdia

Este modelo, também popularmente conhecido como "Codificação Cowboy", se caracteriza pela ausência de uma metodologia de desenvolvimento de software (por isso a figura não conter nada - **Figura 2**). Os sistemas então são construídos na base da informalidade, refletida por uma ausência de qualquer tipo de projeto ou documentação. Cada desenvolvedor realiza suas atividades baseados principalmente em sua experiência pessoal. Neste modelo, o ciclo de desenvolvimento geralmente fica restrito a duas fases: implementação (codificação e demais atividades) e implantação.

Por não ter nenhuma "regra", o modelo Balbúrdia se baseia na pressão política dos superiores de um projeto, em relação ao acréscimo ou ajuste de requisitos do software. Em razão disso, é grande a probabilidade por cobranças à equipe, com relação a prazos e entregas se dará em clima de urgência. Isso ocorrendo, o desgaste (físico e psicológico) ao longo do tempo torna-se inevitável, causando a desmotivação dos profissionais envolvidos no projeto, influenciando diretamente na eficácia da produção.

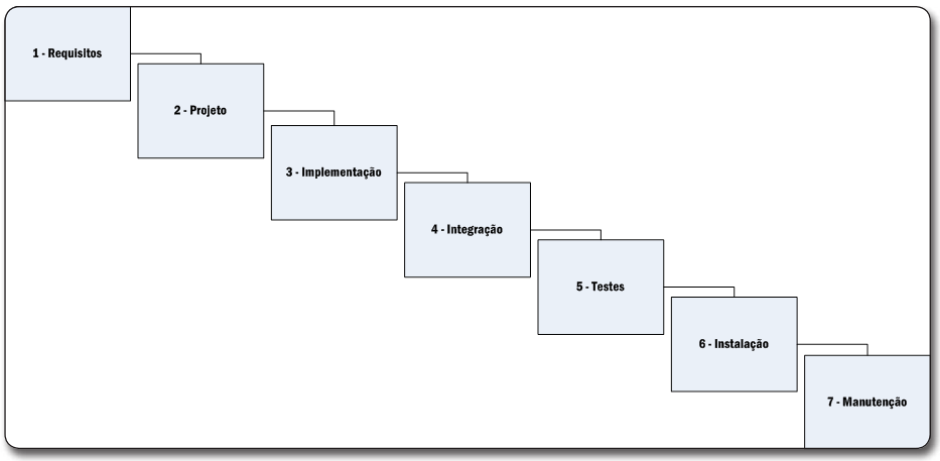


Figura 1. Representação gráfica do modelo em cascata

Figura 2. Representação gráfica do modelo balbúrdia

XP (eXtreme Programming)

XP, acrônimo para o termo em inglês Extreme Programming, é uma metodologia ágil que teve sua aparição inicial em um livro do final da década de 90, intitulado Extreme Programming Explained. Obra esta que foi publicada por Kent Beck, e que se baseia em um trabalho realizado por ele, juntamente com Ron Jeffries e outros profissionais. Ela visa ajudar, principalmente equipes de pequeno e médio porte, na criação de softwares de maior qualidade e em menor tempo. Isto é conseguido através do emprego de ciclos incrementais, ou seja, novas funcionalidades são acrescidas ao projeto a cada iteração, e ajustadas conforme os requisitos do momento.

No que diz respeito à sua aplicação, o XP propõe uma série de práticas que se autocomplementam. Isso quer dizer que os pontos fracos de determinada prática são idealmente compensados pelos pontos fortes de outra. A seguir são exemplificadas algumas dessas principais práticas.

Desenvolvimento orientado a testes

Uma das principais práticas da Extreme Programming é o Test Driven Development, ou desenvolvimento orientado a testes, que prega a criação de testes unitários visando o incremento e a manutenção da qualidade do projeto. Basicamente, neste tipo de desenvolvimento temos três momentos bem definidos. O primeiro é a criação de testes unitários, mesmo não havendo código fonte algum. A seguir, escreve-se o código necessário para que os testes criados funcionem e, finalmente, faz-se uso de refactoring (que será visto a seguir) do código escrito. Entre as vantagens, está a produção de um código mais enxuto e funcional, já que somente o mínimo necessário para a resolução de determinada situação é implementado naquele momento.

Apesar de importante, esta abordagem tende a ser a mais polêmica das práticas propostas, já que acaba por inverter a ordem natural do processo de desenvolvimento tradicional, causando um choque de cultura. Isto pode gerar, por exemplo, alguma resistência por parte da equipe de desenvolvimento, que num primeiro momento não consegue enxergar ganhos com a nova metodologia, intepretando-a como uma simples perda de tempo.

Programação em par

Como o próprio nome sugere, esta prática sugere que a atividade de programação seja realizada por duas pessoas num mesmo computador. O intuito é fazer com que haja uma mescla de experiências entre os envolvidos

e um aumento da qualidade do código fonte gerado. Dessa forma, é recomendado que a dupla seja formada por um profissional iniciante e outro mais maduro, que fará o papel de instrutor. A codificação do software então sempre estará sendo revista por duas pessoas, ou seja, duas mentes trabalhando num mesmo contexto. Entre os ganhos, pode-se citar o aumento da agilidade para se solucionar um problema e a redução da probabilidade da ocorrência de erros.

Refatoração

A prática de Refatoração é um processo de melhoria contínua da programação, de modo que o código esteja sempre em constante evolução, seja no aumento de sua clareza, performance ou estética. Um código refatorado tende a ter uma introdução mínima de erros e ser altamente coeso, contribuindo assim para seu reaproveitamento e evitando possíveis duplicações de código-fonte. A refatoração também é considerada uma prática complementar à prática de desenvolvimento orientado a testes.

Simplicidade

Um dos princípios básicos do XP é a simplicidade, que é a busca pela capacidade em manter um projeto o mais simples possível. Em termos práticos, isso significa que o sistema deve atender tão somente os requisitos necessários àquele momento, exigidos pelo cliente. Num exemplo mais popular, se o cliente solicitou que o software faça a soma de dois números e exiba o resultado, fugiria do escopo fazer com que o sistema realize alguma outra operação matemática ou trabalhe com mais de dois argumentos. Todavia, um erro comum que deve ser evitado é a confusão de conceitos entre código simples e código fácil. O primeiro é o ideal, já que apresenta a solução mais simples para um problema, enquanto que um código fácil de ser implementado nem sempre resulta numa solução ideal e simplista para o projeto.

Jogo de planejamento

No XP, o desenvolvimento é previsto em iterações semanais, assim sendo, a cada início de semana as partes interessadas, cliente e equipe de desenvolvedores, se reúnem a fim de priorizar os requisitos a serem trabalhados na jornada. Esta reunião é denominada Jogo de planejamento ou Planning Game e nela são identificadas as prioridades, por parte do cliente, e consequentemente as estimativas, por parte da equipe. Em vista dessas reuniões semanais, se ganha a flexibilidade necessária, uma vez que, o escopo do projeto estará sendo constantemente reavaliado. Ao final de cada semana, novas funcionalidades estarão sendo postas em produção, ficando a disposição do cliente.

Em um processo mais tradicional, o cliente nem sempre se faz presente neste tipo de reunião, em vista de N fatores, contudo, no jogo de planejamento ele é peça fundamental. Através de sua presença ele consegue, em tempo real, tomar conhecimento do estado atual do projeto e o que está por vir.

Versões pequenas

Esta prática, também conhecida como Small Releases, está intimamente ligada à anterior, uma vez que incentiva a liberação constante de pequenas versões funcionais do software. Isto ajuda no processo de aceitação do projeto por parte do cliente, que pode testar porções do produto que está adquirindo. Em vista dessa liberação constante, obtém-se um contato mais direto com o cliente, recebendo consequentemente mais feedbacks sobre o software que está sendo desenvolvido, em relação a erros e acertos que estão se cometendo.

A partir disso, a estratégia de desenvolvimento pode ser adaptada, a fim de prevenir futuras inconveniências que, numa abordagem mais tradicional, poderiam ser descobertas somente ao final do projeto.

Integração contínua

Integração contínua é a prática onde os trabalhos realizados pelos membros da equipe de desenvolvimento integram freqüentemente seus trabalhos ao projeto, contribuindo assim na detecção antecipada de possíveis erros de integração, produzindo um produto final de maior qualidade.

Em outras palavras, refere-se à capacidade de se manter um conhecimento real sobre a situação que se encontra o projeto através da integração imediata e contínua, de cada nova funcionalidade produzida. Em uma situação contrária, onde a integração ocorre de forma tardia, corre-se o risco do surgimento de possíveis erros e conflitos indesejados.

Teste contínuo

A prática de Teste contínuo é uma consequência da prática de TDD (Test Drive Development) e pode ser dividido em duas partes. A primeira refere-se aos testes unitários, que são testes automatizados, escritos pelos desenvolvedores, conforme prega a prática de Desenvolvimento orientado a testes. Através deles, um desenvolvedor estará, por exemplo, testando determinada funcionalidade de uma classe de negócio do projeto. Assim sendo, este tipo de teste é muito útil à equipe, mas não ao usuário final.

Em vista disso, entra em cena outro tipo de teste denominado Testes de aceitação (ou testes funcionais). Nestes, um usuário final poderá realizá-los, interagindo diretamente com a interface do produto final (software), simulando sua atividade diária com o sistema.

Propriedade coletiva de código

É comum a seguinte situação, perante uma equipe de desenvolvimento envolvida em um projeto: cada membro é intensamente responsável por determinado grupo de código fonte, relacionado a determinada área do sistema. O membro A cuida do código relativo ao módulo 1, o membro B cuida do código do módulo 2 e assim por diante.

Cada membro torna-se praticamente um "proprietário" do seu código, e todo conhecimento e regra do negócio ficam presos a uma determinada pessoa. Isso causa certos transtornos como, por exemplo, quando um membro entra em férias e os demais precisem dar manutenção no código de sua propriedade. Em função desse tipo de situação, o XP apresenta a prática de Propriedade coletiva de código, que estabelece que o código fonte não tem dono, forçando a equipe tomar conhecimento de todas as partes do sistema.

Manutenção do ritmo sustentável

A Manutenção do ritmo sustentável é uma prática que visa manter a motivação da equipe, através de um ambiente de trabalho saudável e um ritmo de trabalho suficiente para mantê-la produtiva. Dessa forma, horas extras são permitidas somente em casos de extrema necessidade e que agregue uma produtividade significativa à execução do projeto.

No entanto, o ritmo ideal é uma jornada de trabalho sem horas extras (Ex.: oito horas diárias / quarenta horas semanais).

Padrão de codificação

Ilustrando uma situação cotidiana, uma equipe de desenvolvimento é composta de várias pessoas, cada qual com seus conceitos e formas de codificação peculiares. Justamente por este motivo, em um mesmo projeto haverá estilos de codificação distintos e seu entendimento por um ou outro membro poderá ser dificultada.

Esta dificuldade ainda pode ser aumentada proporcionalmente ao tamanho da equipe. Uma saída justificável seria o estabelecimento de um padrão de codificação, que é justamente o que esta prática do XP estabelece. No Padrão de codificação, a própria equipe fica responsável por estabelecer as regras relativas à programação, e estas deverão ser essencialmente seguidas por todos os membros. Dessa forma, independente do tamanho da equipe, será produzido um código fonte com estilo único a todo o projeto.

Cliente no local

Conforme já mencionado em outra prática, é interessante a equipe do projeto se manter em contato constante com o cliente, garantindo assim a fluidez ideal do projeto, garantindo que ele atenda as expectativas do negócio. Todo este processo se resume numa prática também conhecida como On-Site Customer que visa, através da proximidade com o cliente (ou de um representante deste), conseguir traduzir mais facilmente os requisitos do negócio para um contexto em que o desenvolvedor possa entender mais facilmente e assim realizar seus trabalhos.

Isso resolve um problema comum, onde se tem de um lado o cliente, detentor do conhecimento sobre o domínio do negócio, e de outro a equipe de desenvolvimento, detentora de conhecimento técnico para construção do software, mas com conhecimento limitado sobre o domínio do negócio.

Cartões de história

Histórias de usuário são especificações fornecidas pelo cliente, em relação aos requisitos que o sistema deverá atender. São utilizadas para se criar estimativas de tempo em uma reunião de planejamento de entrega. Essas informações são descritas em pequenos cartões, denominados cartões de história, que substituem a documentação maçante tradicional. Nesta prática do XP, o cartão tende a apresentar um texto bem simplificado, suficiente apenas para identificar a determinada exigência, facilitando assim o entendimento e a organização. Uma situação comum é a entrega do cartão de história ao desenvolvedor no momento de sua implementação e a devolução deste mesmo cartão para o cliente, ao término da história.

Prototipagem de interface de usuário

A prática de prototipagem de interface de usuário é utilizada como um artefato para análise inicial de um sistema, permitindo uma interação mais real com os envolvidos para tratar dos requisitos que o sistema deverá atender. Dessa forma, com um esboço do sistema em mãos, fica mais fácil decidir em que direção seguir ao se iniciar os trabalhos. Além disso, é possível também mostrar ao cliente a possível interface que ele estará interagindo quando tiver o sistema em produção.

Outro detalhe, é que é imprescindível ter em mente de que não é necessário criar um protótipo para o sistema como um todo, mas sim uma pequena porção mais relevante para o momento, ou até mesmo uma única tela de demonstração.

Reuniões em pé

Com uma intenção diferente das reuniões tradicionais, as reuniões em pé devem ser extremamente curtas, com o foco direcionado às tarefas já realizadas e as tarefas a serem realizadas pela equipe.

Outro diferencial, conforme sua própria nomenclatura sugere, é que este tipo de reunião deve acontecer necessariamente em pé, a fim de se produzir encontros rápidos e que os participantes estejam totalmente focados nos assuntos em pauta.

Scrum

Em meados dos anos 90, surgia o Scrum, um framework ágil para gerenciamento de projetos, cujo conceito principal centrava-se na simplificação da gestão do projeto. Diz-se que o Scrum é o esboço de um processo, baseado em desenvolvimento iterativo. Sua base é formada por um conjunto de três papéis (roles), três documentos e três reuniões.

Papéis

Os três papéis em questão são denominados: Product Owner, Team e Scrum Master.

O Product Owner (ou Proprietário do Produto) é a pessoa representante dos stakeholders no projeto. O exemplo mais tradicional de um stakeholder é o próprio cliente. Assim sendo, o Product Owner tem de garantir, mediante o projeto, que os interesses e necessidades deste elemento sejam fielmente apresentados na forma de requisitos. O Team, ou time, é a equipe de desenvolvimento envolvida. Um time é responsável por desenvolver e testar uma parte entregável do projeto.

Já o Scrum Master é o responsável por garantir que o processo do Scrum flua na organização, fazendo os ajustes necessários para que o método se adapte ao projeto. Ele serve como guia para que a equipe aplique devidamente as práticas do Scrum. Em termos gerenciais, seu principal papel impacta diretamente no sucesso da equipe e do produto, já que ele deve garantir que a equipe faça entregas constantes e de alta qualidade ao invés de sobrepor a qualidade em função de um atraso na conclusão do trabalho.

Documentos

Os três documentos que formam a base do Scrum são: Product Backlog, Sprint Backlog e Sprints Results.

O Product Backlog é o documento onde estão reunidos todos os requisitos do projeto, devidamente priorizados. Uma vez definida esta lista, o desenvolvimento é feito então em pequenas iterações, que no Scrum são chamados de sprints.

Portanto, cada sprint é uma pequena atividade gerenciável, que pode conter desde um design, passando pelo efetivo desenvolvimento e culminando em testes. Em termos de duração, o tempo médio padrão para um sprint é de duas semanas, e o objetivo final de cada etapa é produzir algo entregável ao cliente. Diz-se entregável, pois se espera que este esteja devidamente testado e com a estabilidade suficiente para ser colocado em produção pelo cliente.

Na prática, o início de um sprint é marcado pela escolha dos casos de uso definidos como mais importantes ao negócio, e que poderão ser entregues na iteração atual pela equipe. A partir desta lista de casos de uso, surge então um novo documento no Scrum, denominado Sprint Backlog. O detalhe aqui é que a definição do Sprint Backlog é realizada entre as duas áreas de interesse envolvidas, ou seja, a do cliente e o da equipe. Utilizando as denominações de papéis do Scrum, os interesses do cliente ficam por conta do Product Owner, enquanto que os interesses da equipe são centralizados no Scrum Master e na própria equipe.

Cada item do Sprint Backlog pode ser quebrado em várias pequenas tarefas, que são os trabalhos efetivos a serem realizado dentro da iteração. Finalizando os três documentos do Scrum, cada caso de uso completo pode ser documentado como resultado do sprint (Sprints Results).

Reuniões

As três reuniões base do Scrum são: reunião de planejamento do Sprint, reunião diária e reunião de revisão do Sprint.

A reunião de planejamento de Sprint é o que marca o início de um novo Sprint no Scrum. Nela, Product Owner, Scrum Master e a equipe farão a montagem do Sprint Backlog, baseado nos requisitos definidos no Product Backlog. Neste momento, o Product Owner realiza a priorização dos requisitos mais importantes e, baseado nisso, a equipe faz a negociação com ele para a definição do que poderá ser entregue neste Sprint, baseado nas próprias estimativas.

Durante todo o decorrer do projeto, a cada iteração o Scrum Master encabeça pequenas reuniões diárias, de aproximadamente 15 minutos, para troca de informações entre os envolvidos. Esta atividade define o segundo tipo de reunião do Scrum, a reunião diária. Ainda sobre o tema da reunião, três tópicos são imprescindíveis:

- O que cada membro realizou no dia de ontem;
- O que cada membro pretende realizar no dia de hoje;
- Se há e quais são os empecilhos que estão impedindo a fluidez de suas tarefas.

Os principais objetivos a serem alcançados por essas reuniões diárias é poder se determinar claramente o progresso atingido e, ao mesmo tempo, poder identificar mais rapidamente possíveis falhas ou problemas referentes ao próprio Sprint Backlog e assim fazer os ajustes necessários. Um exemplo seria a adição de novos itens, em vista de um rápido progresso dos trabalhos da equipe. Numa situação oposta, teríamos a movimentação de itens do Sprint Backlog de volta ao Product Backlog, deslocados para serem abordados em uma iteração futura, em razão de uma dificuldade pontual que acabou por atrasar o andamento dos trabalhos da equipe.

Para concluir, ao final de cada Sprint todos os envolvidos (equipe, Scrum Master, Product Owner e stakeholders) se reúnem, com a intenção de analisar os resultados obtidos na iteração que se passou. Supondo uma situação onde se tem várias equipes dispersas, é importante que todos os membros destas tomem conhecimento do que cada uma realizou e assim obter uma visão macro do projeto. Isso pode ser conseguido com pequenas mostras (demos) do que cada uma desenvolveu. Esta reunião é denominada reunião de revisão de Sprint.

Conclusão

No cenário atual do desenvolvimento de software, a complexidade e o tamanho de um projeto influenciam diretamente no tempo disponível para sua construção, que tende a ser reduzido. Com um tempo menor, o surgimento de mudanças significativas durante o processo pode dificultar sua finalização dentro do prazo previsto. Diante deste cenário, as metodologias de desenvolvimento tradicionais não se adaptam de forma eficaz a esta realidade, em vista de sua grande quantidade de processos detalhados, que por si só já exigem um tempo ainda maior. Em outras palavras, corre-se o risco real de se ter rotinas em demasia, relativo à metodologia tradicional e, por conseqüência, gerar atrasos desnecessários.

Todo este cenário teve uma mudança mais impactante com a declaração do Manifesto Ágil, onde foram formalizadas novas formas de se desenvolver software. Desde então, a utilização de metodologias ágeis, tais como os abordados neste artigo (eXtremme Programming (XP) e Scrum) se tornaram uma realidade mais presente nos ambientes de desenvolvimento. Dessa forma, os processos "amarrados" deram lugar a princípios como simplicidade, adaptação às mudanças e software funcional sendo entregue constantemente ao cliente. Agradeço a todos e até a próxima.

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador Delphi há 5 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas além de participar do programa Beta Tester Delphi Prism, da Embarcadero.

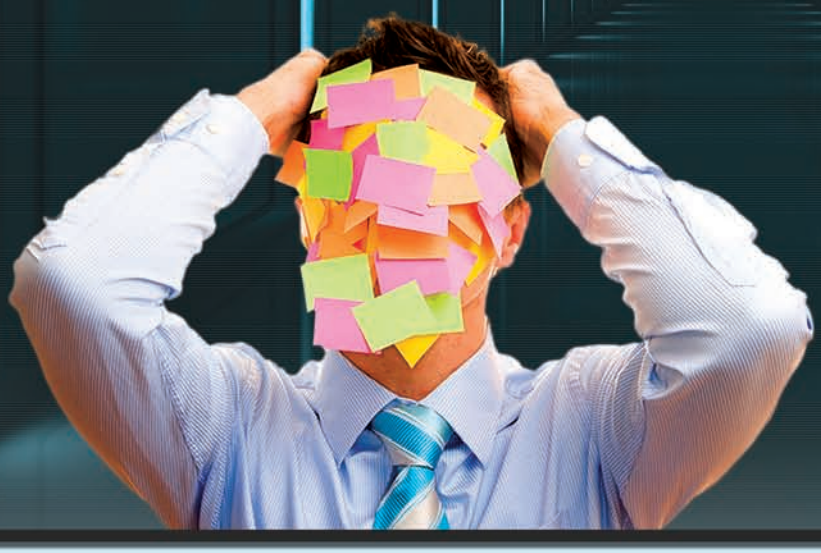
Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Dificuldades em encontrar conteúdo sobre .Net para iniciantes?



A DevMedia simplifica pra você!

easy .net magazine

Easy .Net Magazine:

A primeira revista digital feita especialmente para quem está dando os primeiros passos em .Net!

Você encontrará exemplos e modelos fáceis e práticos, noções introdutórias e conceitos básicos.

Conheça as vantagens:

- Vídeos e apresentações "embutidos" nas páginas;
- Ilustrações, diagramas e modelos apresentados em PowerPoint;
- Entrevistas em vídeo;
- Códigos-fonte que podem ser copiados com CTRL+C/CTRL+V.

A Easy.Net não é feita em PDF ou Flash!
Ela utiliza um aplicativo 100% online.

DevMedia group

Acesse agora a edição Nº 1 totalmente grátis e comprove

www.devmedia.com.br/easynet

Easy.Net Magazine, seu curso inicial de .Net!

Com quantas mentes se revoluciona o mundo?

Campus Party™

6 a 12 de fevereiro de 2012
Anhembi (SP)

Inovação
Cultura Digital
Ciência
Entretenimento Digital

Unimos talento · Criamos futuro

www.campus-party.com.br

@CampusPartyBRA
www.facebook.com/campuspartybrasil